

# **Dokumentation (nach arc42)**

GPA-WS1617-TEAM1

Exported on Okt 30, 2017

## Table of Contents

1.1	Aufgabenstellung .....	5
1.2	Qualitätsziele .....	7
1.3	Stakeholder .....	7
2.1	Technische Randbedingungen.....	9
2.2	Organisatorische Randbedingungen .....	9
2.3	Konventionen.....	10
3.1	Fachlicher Kontext .....	11
3.1.1	Logisches Kontextdiagramm der Buchsuche .....	11
3.1.2	Tabellarische Auflistung der Nachbarsysteme/Benutzer der Buchsuche .....	11
3.2	Technischer- oder Verteilungskontext.....	12
3.2.1	Verteilungsdiagramm des Front- und Backends der Buchsuche .....	12
3.2.2	Technische Interaktion der Buchsuche mit den Nachbarsystemen.....	13
3.3	Externe Schnittstellen .....	14
3.3.1	API Dokumentation der Buchsuche von Team 1 .....	14
3.3.2	API Dokumentation des Bewertung und Empfehlungssystems von Team 2.....	14
3.3.3	API Dokumentation des Kundenverzeichnis von Team 2 .....	14
3.3.4	API Dokumentation der Amazon Product Advertising API .....	14
5.1	Ebene 1 .....	18
5.1.1	BookManagementRestService (BlackBox-Beschreibung) .....	19
5.1.2	BookManagement (Black Box-Beschreibung).....	20
5.1.3	FindBooksUC (Black Box-Beschreibung).....	20
5.1.4	ImportFromAmazonUC (Black Box-Beschreibung) .....	21
5.1.5	ESBookDAO (Black Box-Beschreibung) .....	21
5.1.6	APAClient (Black Box-Beschreibung) .....	22
6.1	Buchsuche .....	23
6.1.1	Sequenzdiagramm der Buchsuche .....	23
6.2	Sequenzdiagramm des Login-Vorgangs und der Buchbewertung .....	24
7.1	Fachliche Strukturen und Modelle .....	25
7.2	Persistenz .....	25
7.3	Benutzungsoberfläche.....	26
7.3.1	Überblick.....	26
7.3.2	GUI Elemente .....	27
7.3.3	Booksearch Architektur .....	30
7.4	Verteilung.....	31
7.4.1	Microservice-Architektur.....	31
7.4.2	Verteilung der Client- und Serverkomponenten im Projekt .....	32
7.5	Konfigurierbarkeit .....	32
7.6	Codegenerierung.....	33
7.6.1	Projektstruktur OASP4J .....	33
7.6.2	Projektstruktur OASP4JS.....	33
7.7	Buildmanagement.....	35
7.7.1	Abstrakter Ablauf der CI/CD-Pipeline.....	35
7.7.2	Versionsverwaltungssystem (VCS).....	35
7.7.3	Build-Management-Tool.....	35
7.7.4	Build des Frontends als eigenständigen Service .....	35
8.1	1. Risiko: Aufwand der Implementierung.....	37
8.2	2. Risiko: Niedrige Softwarequalität durch das Fehlen von Change- und Qualitätsmanagement .....	37
10.1	Verteilungsmatrix Team 1 - Excel Sheet .....	40
11.1	Die Teams.....	41
11.2	Scrumframework.....	41
11.2.1	Scrumteams.....	41
11.2.2	Scrum.....	41
12.1	Erfolge.....	43
12.2	Probleme / Konflikte .....	43
14.1	Technische Probleme .....	46
14.2	Organisatorische Probleme .....	46

**Architekturdokumentation**

Prototyp für Microservices im Online-Buchhandel - Buchsuche und Bücherkatalog

erstellt von

Team 1 - Sebastian Domke, Ilya Sukhov, Nikolai Warkentin

*Template Revision: 6.0 DE (Release Candidate) 19. März 2012*

We acknowledge that this document uses material from the arc 42 architecture template, <http://www.arc42.de>. Created by Dr. Peter Hruschka & Dr. Gernot Starke. For additional contributors see [arc42.de/about/contributors.html](http://arc42.de/about/contributors.html)

**Änderungsübersicht**

Version	Datum	Bearbeiter	Beschreibung

**Basisdokumente**

Dokument	Beschreibung

Inhaltsverzeichnis

**1. Einführung und Ziele**

- 1.1 Aufgabenstellung
- 1.2 Qualitätsziele
- 1.3 Stakeholder

**2. Randbedingungen**

- 2.1 Technische Randbedingungen
- 2.2 Organisatorische Randbedingungen
- 2.3 Konventionen

**3. Kontextabgrenzung**

- 3.1 Fachlicher Kontext
- 3.2 Technischer- oder Verteilungskontext

**4. Lösungsstrategie****5. Bausteinsicht**

- 5.1 Ebene 1
- 5.2 Ebene 2
- 5.3 Ebene 3

**6. Laufzeitsicht**

- 6.1 Laufzeitszenario 1
- 6.2 Laufzeitszenario 2
- 6.3 ...
- 6.4 Laufzeitszenario n

**7. Verteilungssicht**

- 7.1 Infrastruktur Ebene 1
- 7.2 Infrastruktur Ebene 2

## **8. Konzepte**

- 8.1 Fachliche Strukturen und Modelle
- 8.2 Typische Muster und Strukturen
- 8.3 Persistenz
- 8.4 Benutzungsoberfläche
- 8.5 Ergonomie
- 8.6 Ablaufsteuerung
- 8.7 Transaktionsbehandlung
- 8.8 Sessionbehandlung
- 8.9 Sicherheit
- 8.10 Kommunikation und Integration mit anderen IT-Systemen
- 8.11 Verteilung
- 8.12 Plausibilisierung und Validierung
- 8.13 Ausnahme-/Fehlerbehandlung
- 8.14 Management des Systems & Administrierbarkeit
- 8.15 Logging, Protokollierung, Tracing
- 8.16 Geschäftsregeln
- 8.17 Konfigurierbarkeit
- 8.18 Parallelisierung und Threading
- 8.19 Internationalisierung
- 8.20 Migration
- 8.21 Testbarkeit
- 8.22 Skalierung, Clustering
- 8.23 Hochverfügbarkeit

## **9. Entwurfsentscheidungen**

- 9.1 Entwurfsentscheidung 1
- 9.2 Entwurfsentscheidung n

## **10. Qualitätsszenarien**

- 10.1 Qualitätsbaum
- 10.2 Bewertungsszenarien

## **11. Risiken**

## **12. Glossar**

Anmerkung: In der Microsoft-Word-Variante enthält dieses Template Anleitungen und Ausfüllhinweise als „ausgeblendeten Text“. Durch den Befehl „Formate ein-/ausblenden“ können Sie die Anzeige dieser Texte bestimmen.

# 1 Einführung und Ziele

## 1.1 Aufgabenstellung

Ziel dieses Projektes ist es einen lauffähigen Prototyp zu entwickeln, der auf einer Microservice-Architektur basiert. Hierbei wird Wert auf DevOps gelegt und deshalb soll eine robuste CI/CD Pipeline entstehen.

Der Prototyp stellt als Ergebnis einen Online-Buchhandel zur Verfügung, der über mehrere Microservices funktioniert. Das Produkt soll zwei große Funktionsbereiche beinhalten, die in diesem Projekt eigenständige Microservices darstellen. Der erste Part des Prototyps soll eine Buchsuche bereitstellen, der zweite soll einen Bereich für "Meine Bücher" bieten. Diese Dokumentation beschreibt die Microservices "Buchsuche" und "Bücherkatalog".

Der Microservice "Buchsuche" ermöglicht es einem Nutzer ein Buch in unserer Datenbank zu suchen. Hierbei hat er die Möglichkeit nach Titel oder Autor des Buches zu suchen. Als Ergebnis der Suche, bekommt der Nutzer eine Liste mit 20 Elementen mit Angabe von Titel, Autor, Verlag, Erscheinungsjahr, Bewertung und Buchherkunft. Die Suche bietet die Funktionalität der unscharfen Suche (Fuzzy Search), somit reagiert die Suche auch auf Tippfehler des Nutzers und liefert dennoch bestmögliche Ergebnisse.

Für die Buchsuche steht dem System eine Bücherdatenbank mit ~500.000 Elementen zur Verfügung. Diese besteht zum Teil aus einer bestehenden Datenbank, die mit zufällig generierten Büchern erweitert wurde.

Der Microservice der Buchsuche bietet die Funktion Bücher von Amazon zu importieren, falls diese in unserer eigenen Datenbank nicht gefunden wurden. Das heißt, liefert die Suche in unserer Datenbank keine Ergebnisse, wird die Suchanfrage an Amazon geleitet und die dort gefundenen Bücher werden in unsere eigene Datenbank importiert. Ob ein Buch aus unserer Datenbank stammt oder von Amazon importiert wurde, kann an der Spalte "Herkunft" im Suchergebnis entnommen werden.

Hat ein Nutzer Bücher über die Buchsuche gefunden, hat er die Möglichkeit diese im Suchergebnis zu bewerten. In der Spalte "Bewertung" kann er ein beliebiges Buch auf einer Skala von 1 - 5 Sternen bewerten, die Funktion steht dem Nutzer aber nur zur Verfügung, wenn dieser eingeloggt ist. Zusätzlich wird in der Spalte "Bewertung" die durchschnittliche Bewertung jedes Buches angezeigt

<a href="#">UC1</a>	Buchsuche
Akteure	Nutzer
Auslöser	Aufruf der Buchsuche
Aktionsschritte	<ol style="list-style-type: none"> <li>1. Der Nutzer gibt einen Suchbegriff in die Suchzeile ein</li> <li>2. Der Nutzer bestätigt die Suchanfrage</li> <li>3. Das System führt eine unscharfe Suche in Datenbank durch</li> <li>4. Das System gibt 20 Elemente als Suchergebnis auf dem Bildschirm aus</li> </ol>
Ergebnis	Der Nutzer sieht eine Übersicht über die Bücher auf Grundlage der Suchanfrage

UC2	Amazon Bücher importieren
Akteure	Nutzer, Amazon API
Auslöser	Der Nutzer sucht in der Datenbank

UC2	Amazon Bücher importieren
Aktionsschritte	<ol style="list-style-type: none"> <li>1. Der Nutzer führt eine Buchsuche durch</li> <li>2. Das System kann keine passenden Ergebnisse in der Datenbank auf Grundlage des Suchbegriffs finden</li> <li>3. Das System leitet die Suchanfrage an die Amazon API weiter</li> <li>4. Die Amazon API sendet Suchergebnis zum System zurück</li> <li>5. Das System gibt 20 Elemente als Suchergebnis auf dem Bildschirm aus, mit dem Verweis auf Amazon in der Spalte "Herkunft"</li> </ol>
Ergebnis	Der Nutzer erkennt, dass die Ergebnisse der Suche von Amazon stammen

<a href="#">UC3</a>	Bücher bewerten
Akteure	Nutzer, Microservice "Meine Bücher"
Auslöser	Buchsuche liefert Ergebnis
Vorbedingung	Der Nutzer muss eingeloggt sein
Aktionsschritte	<ol style="list-style-type: none"> <li>1. Der Nutzer ermittelt ein Buch, das er bewerten möchte</li> <li>2. Der Nutzer klickt auf die Anzahl von Sternen, mit der er das Buch bewerten möchte</li> <li>3. Das System markiert das Buch mit der entsprechenden Bewertung</li> <li>4. Das System übermittelt die Bewertung an den Microservice von "Meine Bücher"</li> </ol>
Ergebnis	Der Nutzer hat ein Buch mit der gewünschten Bewertung bewertet

UC4	Bücherbewertung einsehen
Akteure	Nutzer, Microservice "Meine Bücher"
Auslöser	Der Nutzer sucht in der Datenbank
Aktionsschritte	<ol style="list-style-type: none"> <li>1. Der Nutzer führt eine Buchsuche durch</li> <li>2. Das System führt die unscharfe Suche in der Datenbank durch</li> <li>3. Das System ruft die Durchschnittsbewertung des Suchergebnis vom Microservice "Meine Bücher" ab</li> <li>4. Das System gibt 20 Elemente als Suchergebnis auf dem Bildschirm aus, die Durchschnittsbewertung wird in der Spalte "Bewertung" angezeigt</li> </ol>
Ergebnis	Der Nutzer erhält die Durchschnittsbewertung der Gefundenen Bücher

## 1.2 Qualitätsziele

Qualitätsmerkmal	Motivation und Erläuterung
System wird auf einem Application Server deployed (Verfügbarkeit)	Damit der Microservice zu jeder Zeit erreichbar ist, wird die aktuellste Version stets auf einem Application Server deployed.  Das entspricht der Projektsituation bei Capgemini und war eine Anforderung an das Projekt.
Das System unterteilt sich in mehrere Microservices (Änderbarkeit)	Um die Funktionalität des Produktes aufzuteilen, wurde das System in vier Microservices unterteilt.  Es ist somit wesentlich einfacher an einzelnen Funktionen des Systems zu arbeiten.
Projekt auf Basis von OASP	Das System basiert auf der Projektstruktur von OASP, somit gelten auch alle Qualitätsziele von OASP für dieses System.

## 1.3 Stakeholder

Stakeholder	Rolle	Ziel / Intention
Stefan Bente	Projektmanager, Auftraggeber	<ul style="list-style-type: none"> <li>• Lernerfolg der Studenten</li> <li>• Praktische Ausführung von Scrum</li> <li>• Erfüllung der Aufgabenstellung: Entwicklung eines Prototyps auf Grundlage der gegebenen Anforderungen</li> <li>• Teamfähigkeit des Teams verbessern</li> </ul>
Axel Burghof	Auftraggeber	<ul style="list-style-type: none"> <li>• Technologievergleich zwischen OASP und JEE im Microservicekontext</li> <li>• Kooperation mit der TH verbessern</li> </ul>
Alex Meier	Infrastruktur-Planung, Konfigurations- & Build-Manager	<ul style="list-style-type: none"> <li>• Betreuung der Entwicklerteams</li> </ul>
Marco Reitano	Infrastruktur-Planung, Konfigurations- & Build-Manager	<ul style="list-style-type: none"> <li>• Betreuung der Entwicklerteams</li> </ul>
Projektteam	Software-Architekt, Entwickler	<ul style="list-style-type: none"> <li>• Erfüllung des Auftrages</li> <li>• Lauffähigen Microservice implementieren</li> </ul>
Microservice Team 2	Verbundene Projekte	<ul style="list-style-type: none"> <li>• Integration der Microservices</li> </ul>

Stakeholder	Rolle	Ziel / Intention

## 2 Randbedingungen

### 2.1 Technische Randbedingungen

Hardware-Vorgaben	Es wurden keine spezielle Anforderungen an die Verfügbarkeit oder die Wiederherstellbarkeit der Hardware vom Auftraggeber formuliert. Die Hardware soll aber eine effiziente Suche über eine große Datenmenge (c.a. 500.000 Einträge) erlauben.
Software-Vorgaben	Die CD-Pipeline soll nicht Docker für die Containerverwaltung verwenden. Die *.war-Dateien sollen auf einem gemeinsamen Applicationsserver deployt werden. Als Applicationsserver soll Tomcat zum Einsatz kommen. Als webbasiertes Software-System zur kontinuierlichen Integration von Komponenten zu einem Anwendungsprogramm soll Jenkins eingesetzt werden.
Weboberfläche	Die Weboberfläche soll bei der Verwendung von <a href="#">OASP4JS</a> entwickelt werden. Es wurden keine speziellen Anforderungen an das UI oder in Hinblick auf Usability vom Auftraggeber gestellt.
Vorgaben des Systembetriebs	Das Betriebssystem soll entweder als Open Source oder über das Dreamspark-Programm verfügbar und in der Lage sein als Webserver zu agieren.
Programmervorgaben	Implementierung soll in Java erfolgen. Der Microservice soll bei der Verwendung von dem <a href="#">OASP-Framework</a> und unter Einhaltung von den, im Framework festgelegten, Standards und der Referenzarchitektur entwickelt werden.
Datenstruktur	Eine Buch-Entität soll durch folgende Eigenschaften beschrieben werden: Titel, Autor, Verlag, Erscheinungsjahr.

### 2.2 Organisatorische Randbedingungen

Organisation und Struktur	Das Team besteht aus <a href="#">Sebastian Domke</a> , <a href="#">Nikolai Warkentin</a> und <a href="#">Ilya Sukhov</a> unterstützt von wissenschaftlichen Mitarbeiter der TH Köln <a href="#">Marco Reitano</a> und <a href="#">Alex Maier</a> . Weitere Stakeholder sind dem Kapitel <a href="#">Stakeholder</a> zu entnehmen.
Zeitplan	Beginn der Entwicklung in Oktober 2016. Jede 4 Wochen soll ein lauffähiges Inkrement des Prototyps geliefert werden. Fertigstellung der Version 1.0 nach 16 Wochen der Entwicklung.
Organisatorische Standards	Die Software soll inkrementell und iterativ unter Verwendung des Scrum-Frameworks und den darin festgelegten Regeln und Prozeduren entwickelt werden. Zur Dokumentation der Architektur kommt arc42 zum Einsatz. Die Projektdokumentation soll in Atlassian Confluence Wiki erstellt werden. Die Verwaltung von Arbeitspaketen soll durch die Verwendung von Atlassian JIRA erfolgen.
Entwicklungswerkzeuge	Erstellung der Java-Quelltexte in Eclipse oder IntelliJ. Die Software muss jedoch auch, allein mit Gradle, also ohne IDE baubar sein, um eine Automatisierung des Deployment zu ermöglichen. Als Version

	Control System soll Git zum Einsatz kommen. Die Verwaltung von Git-Repositories soll durch den Einsatz von GitLab erfolgen.
--	---

## 2.3 Konventionen

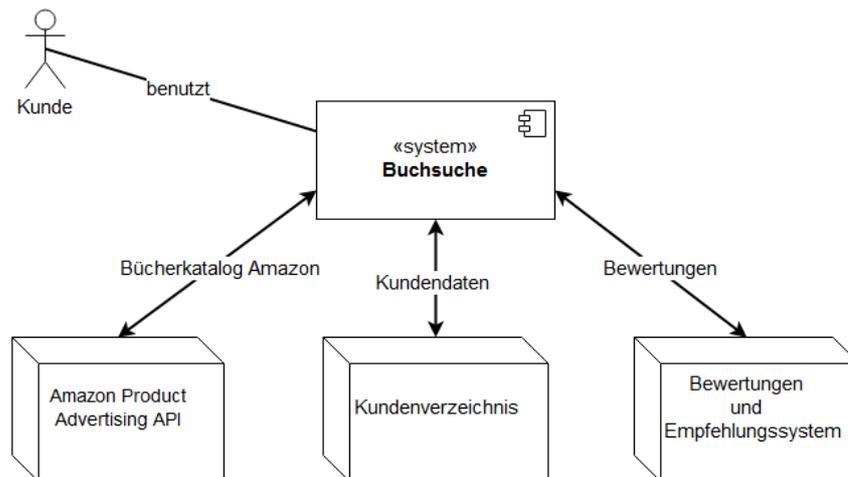
Kodierrichtlinien für Java	<p>Die Einhaltung von den, durch die OASP-Community festgelegten, <a href="#">Kodierrichtlinien</a>.</p> <p>Die Einhaltung von den, durch die OASP definierten, <a href="#">Webbaplikationsarchitektur</a> (Komponenten, Modules, Standards)</p>
Sprache	<p>Verwendung englischer Bezeichner für Klassen, Methoden etc. und im Java-Quelltext.</p> <p>Die arc42-Dokumentation soll in deutscher Sprache erstellt werden. Die Microservice API-Dokumentation kann in englischer Sprache erstellt werden.</p> <p>Es soll nur Englisch innerhalb der Webanwendung bzw. Microservice unterstützt werden.</p>
Versions- und Konfigurationsmanagement	Die einzelnen Scrum User Stories sollen innerhalb von <a href="#">Feature Branches</a> implementiert werden.

### 3 Kontextabgrenzung

Die folgenden Unterkapitel zeigen die Einbettung unseres Systems in seine Umgebung.

#### 3.1 Fachlicher Kontext

##### 3.1.1 Logisches Kontextdiagramm der Buchsuche



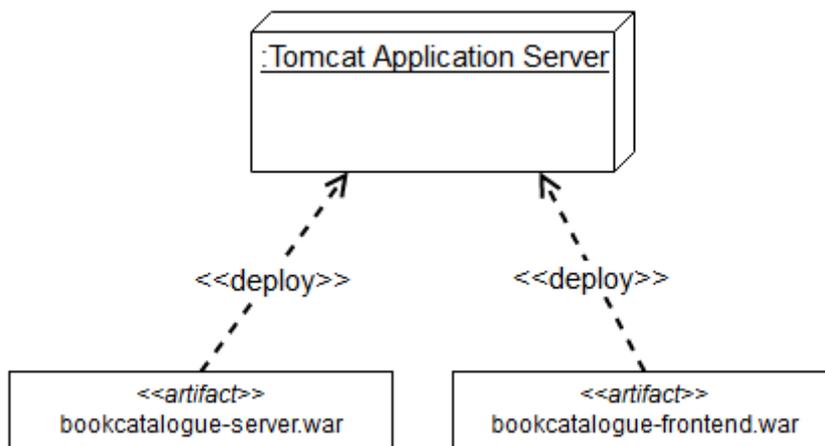
##### 3.1.2 Tabellarische Auflistung der Nachbarsysteme/Benutzer der Buchsuche

Buchsuche			
Nachbarsystem/Benutzer	Beschreibung	Input	Output
Kunde	Benutzer der Buchsuche. Dieser kann über das System nach Büchern (Titel/Autor/Titel und Autor) suchen, die Durchschnittsbewertungen der gefunden Bücher einsehen und eigene Bewertungen abgeben.	Suchanfrage Bewertung	Suchergebnis
Amazon Product Advertising API	Schnittstelle zum Produktkatalog von Amazon. Über die API können Artikel des Onlineangebots von Amazon gesucht und abgefragt werden.	REST request	XML response
Kundenverzeichnis	Verzeichnis aller Kunden. Das Verzeichnis enthält alle Kundendaten (Name und	Benutzeranmeldung	Benutzerinformation

Buchsuche			
	E-Mail-Adresse) und dient dabei zur Anmeldung eines Kunden  und der Zuordnung von Aktionen im System.		
Bewertungen und Empfehlungssystem	Das System dient dazu Bewertungen des Kunden für ein Buch zu speichern, Durchschnittsbewertungen abzufragen und dem Kunden Empfehlungen auf Basis seiner bereits bewerteten Bücher anzubieten.	Kundenbewertung von Büchern	Durchschnittsbewertung von Büchern

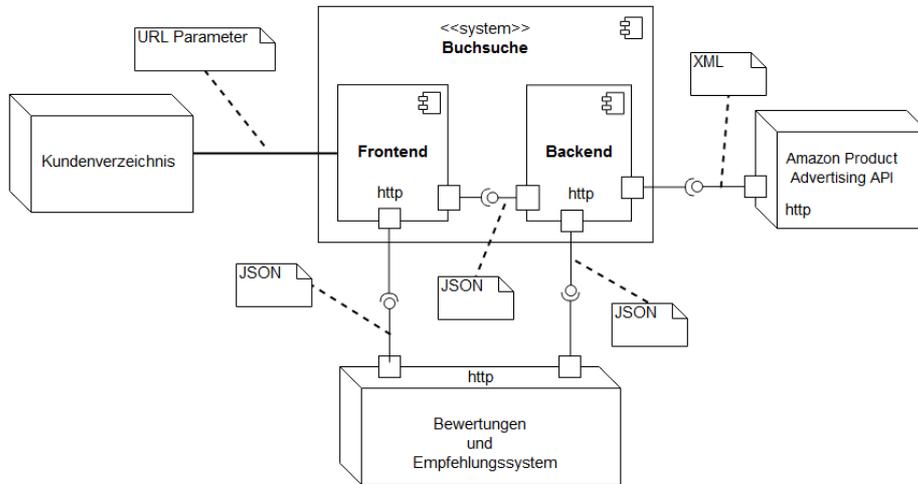
## 3.2 Technischer- oder Verteilungskontext

### 3.2.1 Verteilungsdiagramm des Front- und Backends der Buchsuche



Eine detaillierte Beschreibung des Deployments der Front- und Backendkomponente im Rahmend der CI/CD-Pipeline erfolgt im Abschnitt [Buildmanagement](#).

### 3.2.2 Technische Interaktion der Buchsuche mit den Nachbarsystemen



Mappingtabelle - Buchsuche und Nachbarsysteme			
Kanal	Beschreibung	Input	Output
search API	Suche im Bücherkatalog. Anbindung des Frontends der Buchsuche an das Backend.	search query (String) searchBy (String) count (Number)	search result (JSON)
books API	Abfrage von Büchern aus dem Bücherkatalog. Anbindung des Bewertungs und Empfehlungssystems an das Backend der Buchsuche.	bookIDs (String)	books (JSON)
ratings API	Abfrage der Durchschnittsbewertung von Büchern. Anbindung des Frontends der Buchsuche an das Bewertungs und Empfehlungssystem.	bookIDs (String)	ratings (JSON)
rate API	Bewertung von Büchern. Anbindung des Frontends der Buchsuche an das Bewertungs und Empfehlungssystem.	bookID (String) rating (Integer) userID (JSON)	ratings (JSON)
Amazon Product Advertising API	Ermöglicht die Nutzung des Onlinekatalogs von Amazon. Anbindung des Backends der Buchsuche an den Bücherkatalog von Amazon.	signedRequest (String)	search result (XML)
customer	Ermöglicht die Übergabe der UserID des angemeldeten Benutzers. Anbindung des Frontends der Buchsuche an das Kundenverzeichnis.	source url (String)	userID (URL Parameter)

## 3.3 Externe Schnittstellen

### 3.3.1 API Dokumentation der Buchsuche von Team 1



### 3.3.2 API Dokumentation des Bewertung und Empfehlungssystems von Team 2

<http://fsygs15.gm.fh-koeln.de:8280/swagger/?url=http://fsygs15.gm.fh-koeln.de:8280/recommendation-ms/swagger.json>

### 3.3.3 API Dokumentation des Kundenverzeichnis von Team 2

<http://fsygs15.gm.fh-koeln.de:8280/swagger/?url=http://fsygs15.gm.fh-koeln.de:8280/customers-ms/swagger.json>

### 3.3.4 API Dokumentation der Amazon Product Advertising API

<https://aws.amazon.com/archives/Product-Advertising-API/8967000559514506>

## 4 Lösungsstrategie

Epic	User Story	Task	Beschreibung
<a href="#">GPAMSONE-1:</a> Bücherkatalog	<a href="#">GPAMSONE-24</a> Als Benutzer möchte ich die Funktionalität einer Suche gegeben haben, um ein Buch in der Datenbank zu finden.	<a href="#">Such-API</a>	Die Funktionalität der Buchsuche wird von der Elastic Search Engine übernommen. Elastic Search bietet eine schnelle und unscharfe Such an und kann über Plugins erweitert und nach eigenen Bedürfnissen angepasst werden. Da diese Engine die Anforderungen der Unscharfen Suche erfüllt und weit verbreitet ist wurde sich in diesem Projekt dafür entschieden. Die Elastic Search Komponente wird als Teil des Backends angesehen, es wäre aber auch möglich sie als eigenständige Komponente zu betrachten.
		<a href="#">Suchalgorithmus</a>	
		<a href="#">Konfiguration von ES</a>	
		<a href="#">Konfiguration von Spring Data ES</a>	
	<a href="#">GPAMSONE-6</a> Als Benutzer möchte ich von außen auf einen Bücherkatalog zugreifen, um mir Informationen über ein Buch aufrufen zu lassen.	<a href="#">Datenbankverwaltungssystem auswählen</a>	Zur Datenhaltung wurde eine MySQL-Datenbank verwendet, die jedoch aus Performancegründen durch die Elastic Search eigene Datenbank ersetzt wurde. Außerdem wäre die Verwaltung zweier Datenbanken redundant und würde unnötig Ressourcen verwenden.
	<a href="#">Datenbankschema definieren</a>		
	<a href="#">Datenbank mit Testdaten füllen</a>		
	<a href="#">GPAMSONE-18</a> Als Auftraggeber möchte ich einen Dummy mit Restschnittstelle haben der deploybar ist, um die Funktionsfähigkeit der Deployment-Pipeline zu testen.	<a href="#">Rest-Service implementieren</a>	Rudimentäre CI/CD Pipeline mit Rest-Schnittstelle implementiert. Es wurden Dummydaten erzeugt um die Schnittstelle zu testen.
	<a href="#">Json spezifizieren</a>		
	<a href="#">Dummydaten bereitstellen</a>		
<a href="#">GPAMSONE-2:</a> Buchsuche	<a href="#">GPAMSONE-27</a> Als Auftraggeber möchte ich, ein lauffähiges Frontend für die Buchsuche als eigenständigen Service haben, um darauf aufbauend zu einem späteren	<a href="#">Einrichtung der CD-Pipeline für das Frontend</a>	Für das Frontend wurde eine eigene CD-Pipeline aufgebaut, damit das Frontend unabhängig vom Backend als eigenständiger Microservice (MS) deployed werden kann. Es wurde sich für diese
		<a href="#">Konfiguration des Frontend als eigenständigen Service</a>	

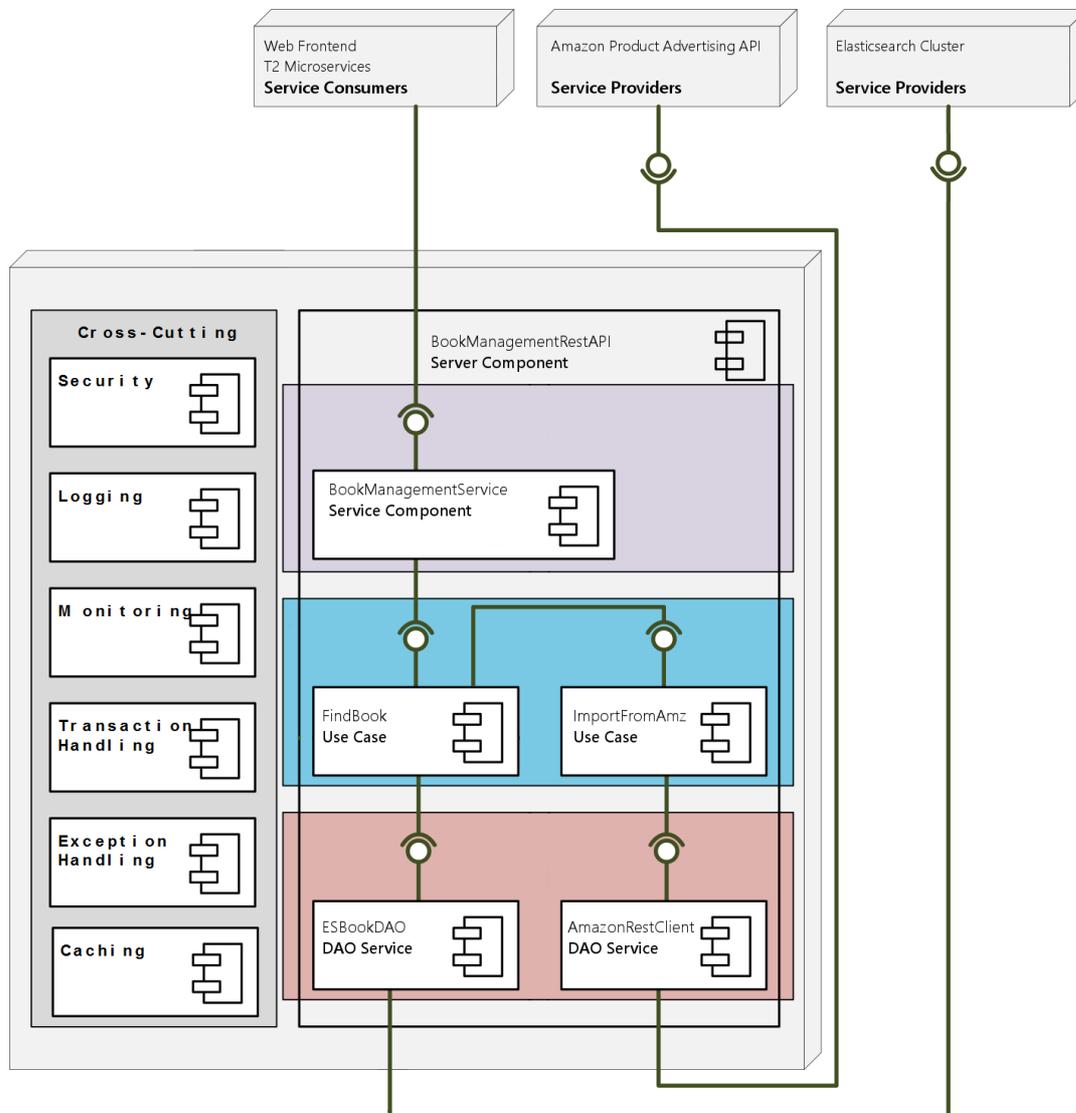
Epic	User Story	Task	Beschreibung
	Zeitpunkt das Frontend selbst entwickeln zu können.		Lösung entschieden, da es sich um ein Microservice Projekt handelt und sich das Frontend als eigenen Microservice unabhängig vom Backend verwalten lässt.
	<a href="#">GPAMSONE-29</a> Als Benutzer möchte ich die Möglichkeit haben die durchschnittliche Bewertung der Bücher in der Ergebnisliste der Buchsuche einzusehen, um die Qualität der Bücher zu sehen.	<a href="#">Integration des Bewertungs-MS</a> <a href="#">Ergebnisliste der Buchsuche um die durchschnittliche Bewertung erweitern</a>	Der MS Meine-Bücher wurde in unser Frontend integriert, um die Durchschnittsbewertung der Bücher abrufen und anzeigen zu können.
	<a href="#">GPAMSONE-28</a> Als Benutzer möchte ich die Möglichkeit haben Bücher in der Ergebnisliste der Buchsuche bewerten zu können, damit das Buch zu "Meine Bücher" hinzugefügt wird.	<a href="#">AngularJS-Komponente für die Bewertung aussuchen</a> <a href="#">Ergebnisliste der Buchsuche um eine Bewertungsfunktion erweitern</a> <a href="#">Integration des Bewertungs-MS</a>	Für die Anzeige der Bewertung (Sterne) im Frontend wurde eine Opensource Directive für AngularJS verwendet. Somit war es möglich eine Bewertung abgeben zu können und diese an die API vom MS Meine-Bücher zu übergeben. Da es bereits eine Opensource lösung für das Bewertungssystem gab, wurde darauf verzichtet
	<a href="#">GPAMSONE-41</a> Als Auftraggeber möchte ich, dass Benutzer Bücher erst nach Anmeldung bewerten können, damit die Bewertung dem Benutzer zugeordnet werden kann.	<a href="#">Integration von User MS</a> <a href="#">UI anpassen</a> <a href="#">Bewertungs-MS vollständig integrieren</a>	Um zu registrieren, ob ein Benutzer angemeldet ist oder nicht, wird die UserID im Frontend verwaltet. Sie wird beim Anmelden über den Login MS an unser Frontend übergeben und ausgelesen. Somit ist eine primitive Benutzerverwaltung vorhanden. Eine ausgereifere Userverwaltung zu implementieren wurde nicht umgesetzt, da dies für dieses Projekt nicht gefordert war.
		<a href="#">Einbindung der Amazon API</a>	

Epic	User Story	Task	Beschreibung
	<a href="#">GPAMSONE-5</a> Als Auftraggeber möchte ich, dass wenn kein passendes Buch gefunden wurde Vorschläge von Amazon angezeigt werden und in den Bücherkatalog importiert werden, um die Suchergebnisse zu erweitern.	<a href="#">Datenbankschema um findBy Amazon erweitern</a> <a href="#">Bei Amazon gefundene Bücher automatisch in DB übertragen</a> <a href="#">Team 2 mitteilen, dass Body um Spalte erweitert wurde</a>	Um den Funktionsumfang unseres MS zu erweitern, wurde die Amazon API in das System eingebunden. So kann die System eigene Datenbank um die Bücher aus der Amazon Datenbank erweitert werden. Diese optionale Funktionalität wurde dem System hinzugefügt, da noch Ressourcen für die Implementierung verfügbar waren.
	<a href="#">GPAMSONE-39</a> Als Benutzer möchte ich, bei der Benutzung der Buchsuche feinere Resultate angezeigt bekommen, um schneller die gesuchten Bücher zu finden.	<a href="#">Konfigurieren von Elastic Search Suchanfragen</a>	Feinjustierung von Elasticsearch. Elasticsearch bietet einen großen Funktionsumfang und viele Einstellmöglichkeiten für die Suche.
	<a href="#">GPAMSONE-4</a> Als Benutzer möchte ich über ein Suchformular nach Büchern in Form von Titel oder Autor suchen, um das gewünschte Buch zu finden.	<a href="#">UI Suchfeld</a> <a href="#">UI Suchergebnisanzeige</a>	Suchfunktion um Suchoptionen erweitert. Suche nach Titel, Autor, Titel und Autor möglich.
<a href="#">GPAMSONE-7:</a> Technologien	<a href="#">GPAMSONE-8</a> Als Entwickler möchte ich mich mit der Deployment-Pipeline auseinandersetzen, um dem Kunden das Betreiben von DevOps zu ermöglichen.	<a href="#">GitLab Repository für unser Projekt anlegen</a> <a href="#">Jenkins konfigurieren</a> <a href="#">Applicationserver auswählen</a> <a href="#">Applicationserver aufsetzen</a> <a href="#">Aufsetzen und Einrichten der Entwicklungsumgebung</a>	Aufbau der CI/CD Pipeline. Konfiguration aller Komponenten (GitLab, Jenkins, Tomcat). Die gewählten Technologien waren zu größten Teil durch die Serverlandschaft vorgegeben und wurden deshalb so übernommen.

## 5 Bausteinsicht

### 5.1 Ebene 1

Die folgende Abbildung zeigt die Hauptbausteine unseres Systems und deren Abhängigkeiten:



Die Module einer höheren Ebene sind von den Modulen auf einer niedrigeren Ebene abhängig. Die Service-Komponenten wie Security, Logging, Monitoring, Transaktion und Exception Handling, Caching sind vom OASP bereitgestellt. Es werden nur die Service-Komponenten eingebunden, die bei der Abdeckung von Anforderungen notwendig sind. In unserem Fall haben wir Logging und Exception Handling Komponenten intensiv benutzt. Die horizontale Trennung von Komponenten in Service, Logic und Data Access Layers ist ebenso von OASP vorgegeben. Die Komponenten aus Service Layer stellen REST-Schnittstellen nach außen bereit, die von externen Systemen (Webapplikation oder die Microservices vom Team 2) genutzt werden können. Die Komponenten aus Data Access Layer greifen auf Microservice-externe Dienste wie [Amazon Product Advertising API](#) und [ElasticSearch](#) cluster (in unserem Fall auf der gleichen Maschine wie Webserver installiert).

Subsystem	Kurzbeschreibung
BookManagementRestService	Realisiert die Kommunikation mit Clients über eine REST-Schnittstelle
BookManagement	Use cases Orchestrierung.
FindBookUC	<i>Find book</i> use case ist für die Suche in Book Repository zuständig
ImportFromAmazonUC	<i>Import from Amazon</i> use case ist für den Import der Bücher aus Amazon zuständig
ESBookDAO	Ist für den Zugriff auf die Daten in ElasticSearch zuständig
APAClient	Ist für den Zugriff auf die <a href="#">Amazon Product Advertising API</a> zuständig

### 5.1.1 BookManagementRestService (BlackBox-Beschreibung)

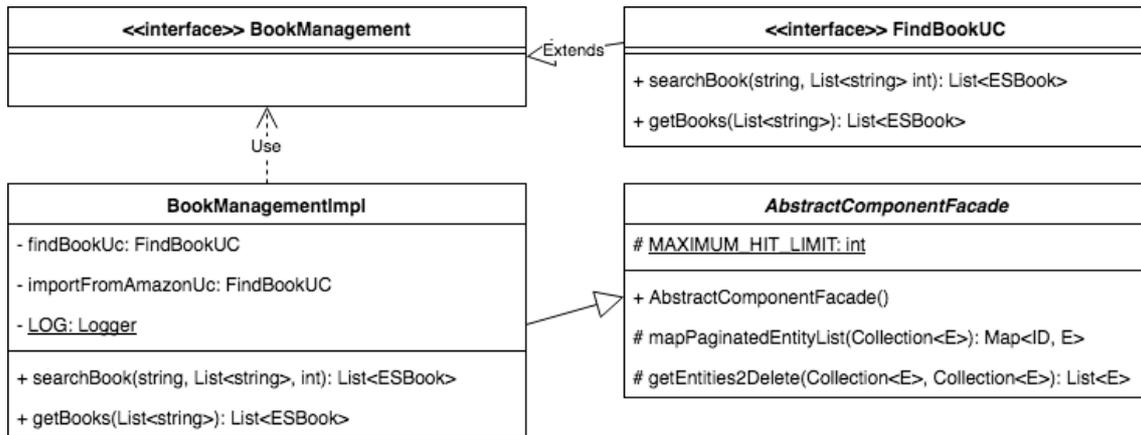
<b>BookManagementRestService</b>
- LOG: Logger - bookManagement: BookManagementUC
+ searchBook(UriInfo): List<ESBook> + getBooks(UriInfo): List<ESBook>

Methode	Kurzbeschreibung
searchBook	Validiert die mit einer Query an das Service übergebene Parameter und startet eine Suche nach passenden Büchern.
getBooks	Validiert die mit einer Query an das Service übergebene Parameter und gibt die, innerhalb der Query, angeforderten Bücher zurück.

Ablageort/Datei:

`de.th.microservice.bookcatalogue.bookcatalogue.service.api.rest`

### 5.1.2 BookManagement (Black Box-Beschreibung)

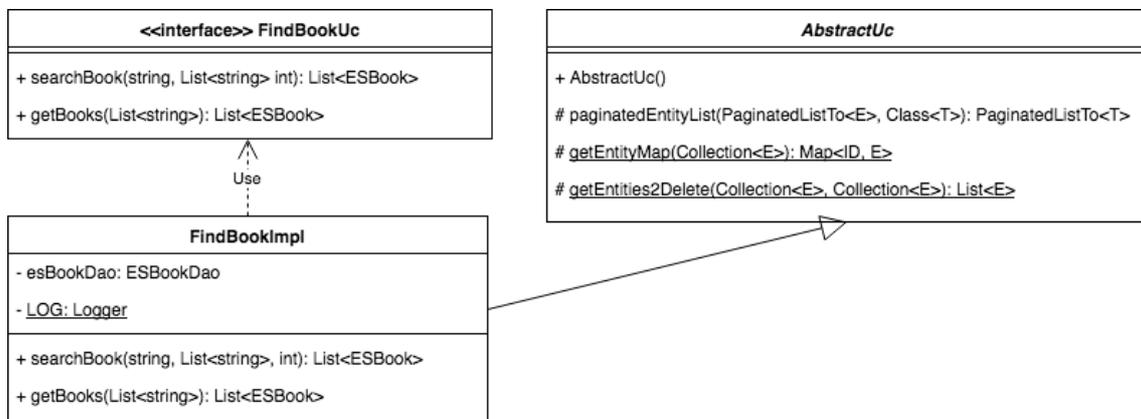


Methode	Kurzbeschreibung
searchBook	Startet eine Suche nach Büchern unter Berücksichtigung von definierten Suchparametern
getBooks	Nimmt eine Liste von Bücher-ID's als Input und liefert entsprechende Buch-Entitäten

Ablageort/Datei:

de.th.microservice.bookcatalogue.bookcatalogue.logic.impl

### 5.1.3 FindBooksUC (Black Box-Beschreibung)

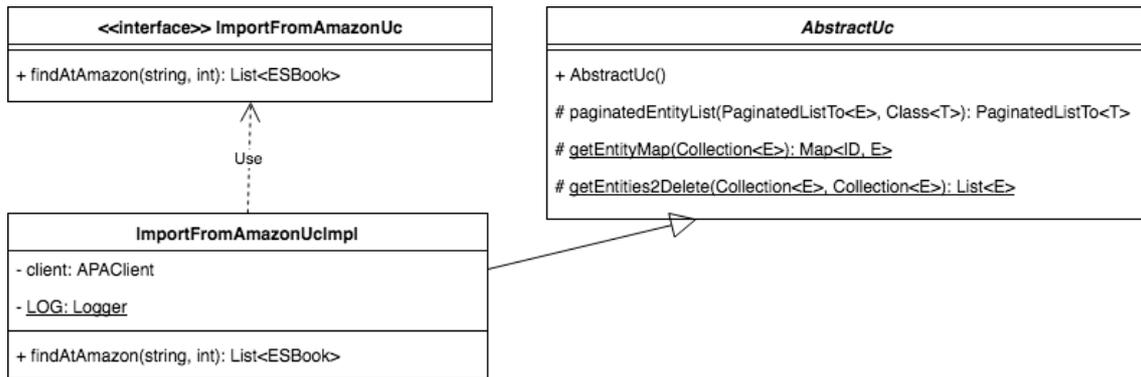


Methode	Kurzbeschreibung
searchBook	Startet eine Suche nach Büchern unter Berücksichtigung von definierten Suchparametern
getBooks	Nimmt eine Liste von Bücher-ID's als Input und liefert entsprechende Buch-Entitäten

Ablageort/Datei:

de.th.microservice.bookcatalogue.bookcatalogue.logic.impl.usecase

### 5.1.4 ImportFromAmazonUC (Black Box-Beschreibung)

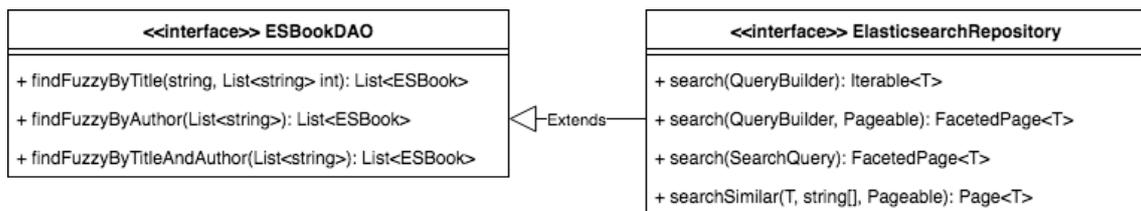


Methode	Kurzbeschreibung
findAtAmazon	Startet eine Suche nach Bücher bei Amazon unter Berücksichtigung von definierten Suchparametern

Ablageort/Datei:

de.th.microservice.bookcatalogue.bookcatalogue.logic.impl.usecase

### 5.1.5 ESBookDAO (Black Box-Beschreibung)



Methode	Kurzbeschreibung
findFuzzyByTitle	Startet eine Suche nach Büchern bei ElasticSearch Repository über die "Titel"-Entitätsfeld
findFuzzyByAuthor	Startet eine Suche nach Büchern bei ElasticSearch Repository über die "Autor"-Entitätsfeld
findFuzzyByTitleAndAuthor	Startet eine Suche nach Büchern bei ElasticSearch Repository über die "Autor"- und "Titel"-Entitätsfelder

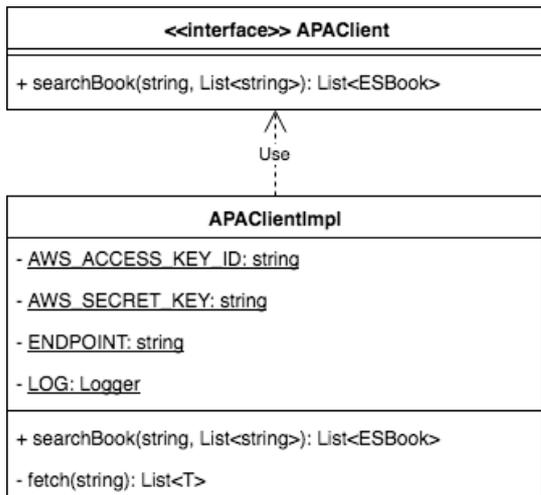
Diese Schnittstelle wird nicht direkt instantiiert. Jede Methode der Schnittstelle ist mit einer @Query-Annotation versehen, die bei der Verwendung von ElasticSearch Query DSL zusammengestellt wurde.

Die eigentliche Implementierung der Schnittstelle wird dadurch durch das Spring ElasticSearch Data Modul generiert.

Ablageort/Datei:

de.th.microservice.bookcatalogue.bookcatalogue.dataaccess.api.dao

### 5.1.6 APAClient (Black Box-Beschreibung)



Methode	Kurzbeschreibung
searchBook	Startet eine Suche nach Büchern bei Amazon unter Berücksichtigung von definierten Suchparametern
fetch	Führt eine Anfrage aus und parst die Antwort

Ablageort/Datei:

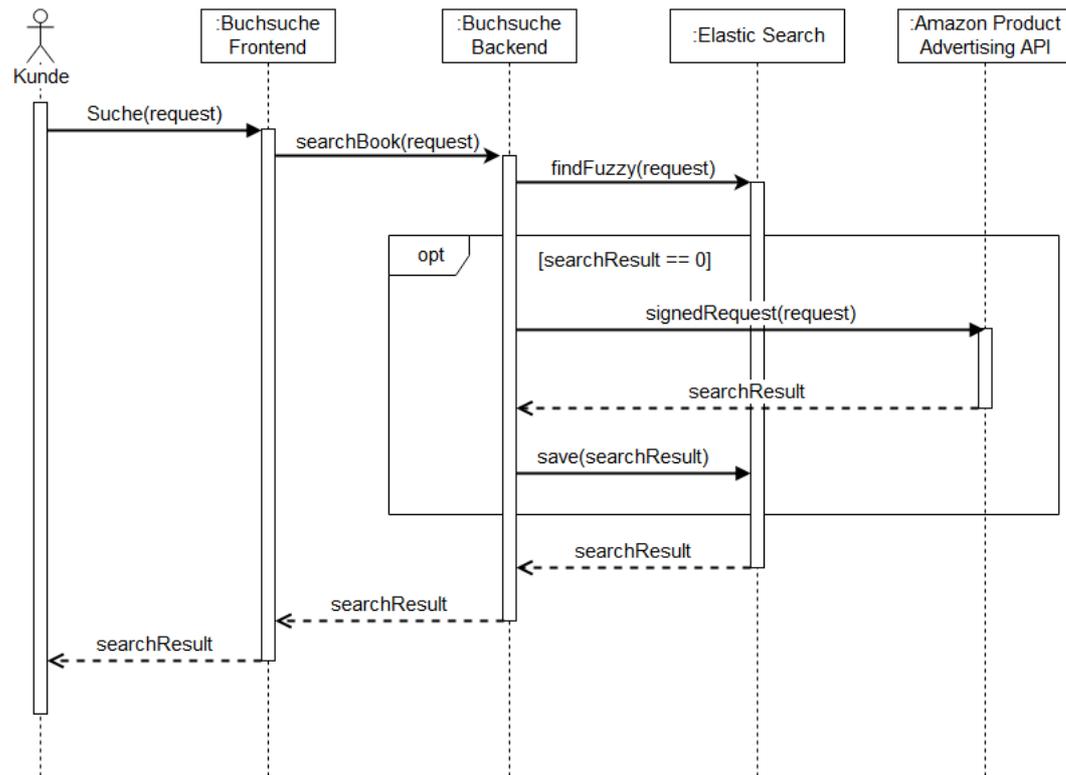
de.th.microservice.bookcatalogue.bookcatalogue.dataaccess.impl.dao

## 6 Laufzeitsicht

Laufzeitsicht der Buchsuche und des Loginvorgangs in Verbindung mit einer Buchbewertung.

### 6.1 Buchsuche

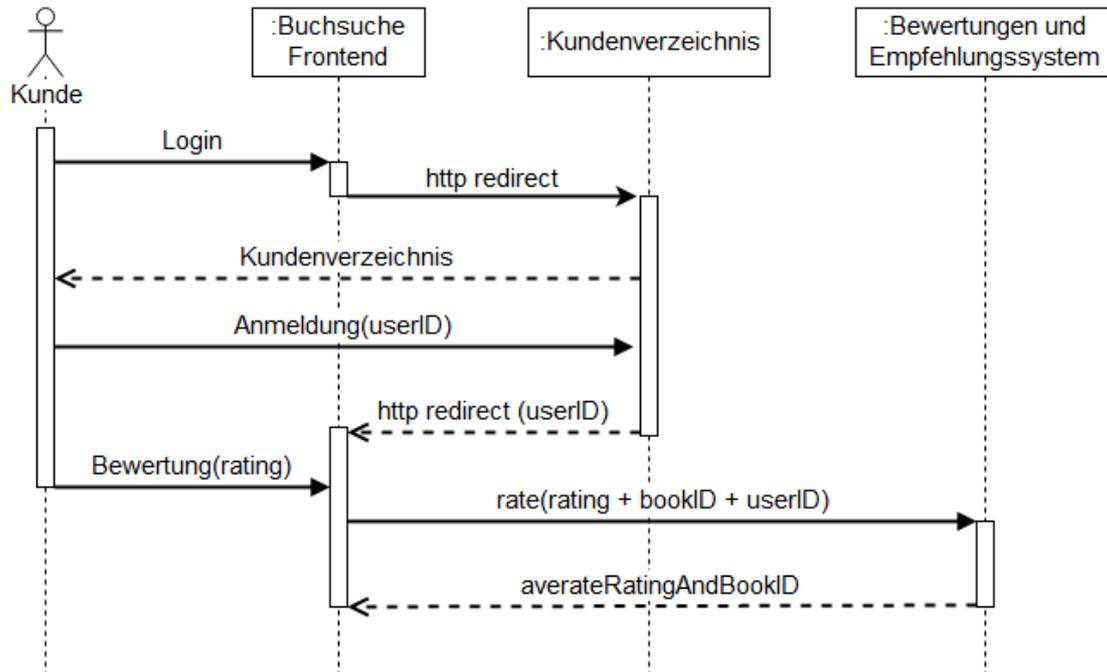
#### 6.1.1 Sequenzdiagramm der Buchsuche



Das Sequenzdiagramm der Buchsuche basiert auf dem Use Case 1 - Buchsuche aus dem Abschnitt [Aufgabenstellung](#).

#### 6.1.2

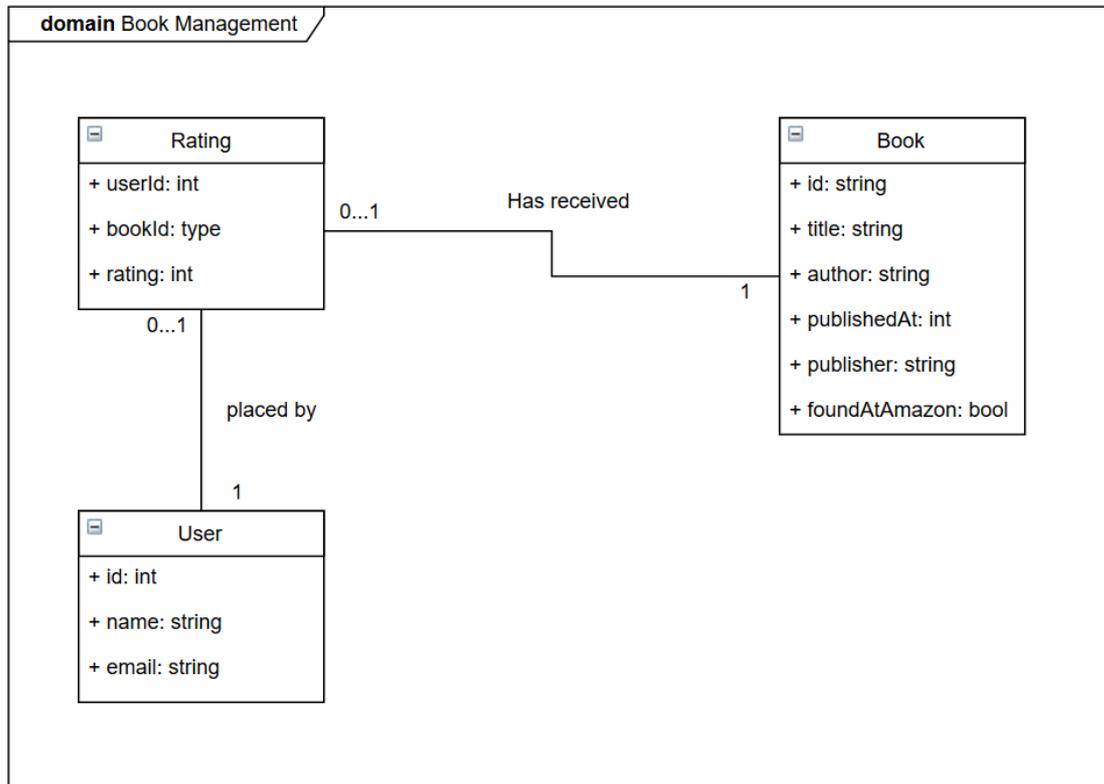
## 6.2 Sequenzdiagramm des Login-Vorgangs und der Buchbewertung



Das Sequenzdiagramm des Login-Vorgangs und der Buchbewertung basiert auf dem Use Case 3 - Bücher bewerten aus dem Abschnitt [Aufgabenstellung](#).

## 7 Konzepte

### 7.1 Fachliche Strukturen und Modelle



### 7.2 Persistenz

Für die Abbildung von Java-Objekten auf die Datensätze einer Datenbank wird die [Java Persistence API](#) (JPA) verwendet. Obwohl OASP empfiehlt Hibernate als JPA-Implementierung zu verwenden, wird im Projekt eine andere Implementierung namens Spring Data eingesetzt. Der Grund dafür ist die Verwendung von NoSQL Data Store namens [ElasticSearch](#).

ElasticSearch wird eingesetzt um die Anforderung bezüglich der Geschwindigkeit der unscharfen Suche bei großen Datenmengen abzudecken. Der wesentliche Nachteil bei der Verwendung von NoSQL-Datenbanken ist die Notwendigkeit die Daten redundant zu speichern, da die Datensätze in 1. Normalform gespeichert werden.

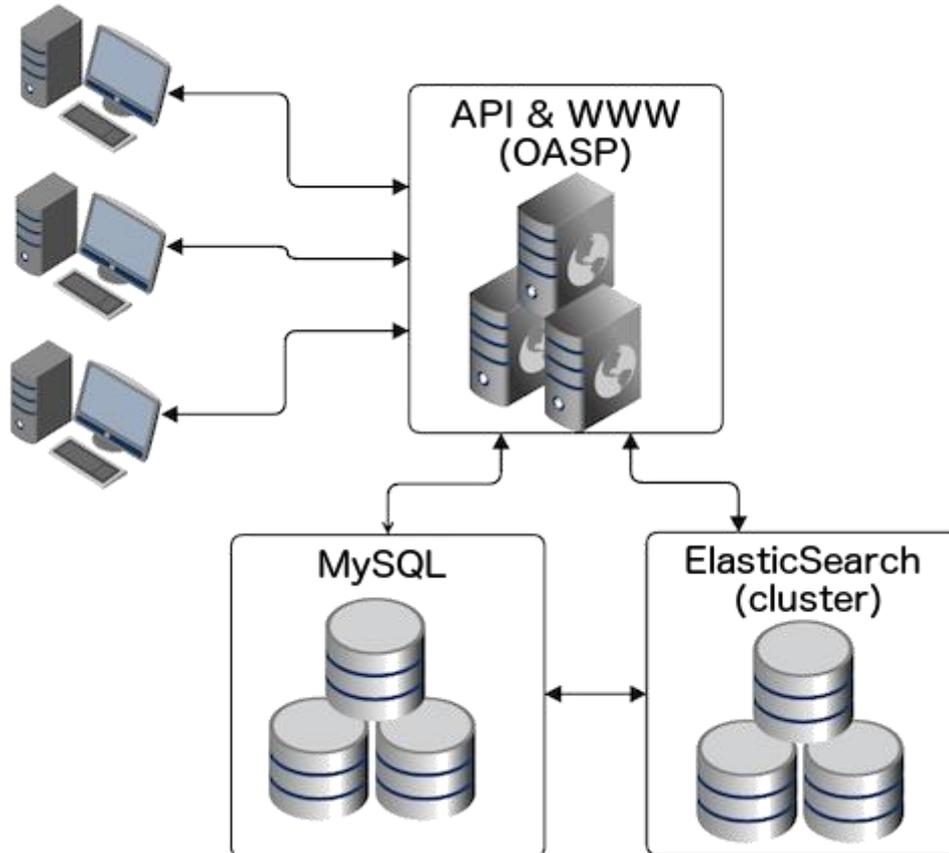
Es lohnt sich aber in unserem Fall alle Informationen in ElasticSearch zu speichern, da die Datenstruktur beschrieben durch nur eine Entität ziemlich trivial ist und der *Data-Overhead* nicht so drastisch wird. Ein weiterer Vorteil von ElasticSearch ist die Unterstützung einer Vielzahl von Fuzzy Search Algorithmen *out-of-the-box*. Allerdings hängen beide Frameworks, OASP und Spring Data, von dem Spring Boot Framework ab. Da das OASP nur eine alte Version von Spring Boot unterstützt, sind wir darauf angewiesen ebenso eine ältere Version von Spring Data ElasticSearch (1.3.4) einzusetzen, was wiederum dazu führt, dass wir nur die Version 1.7.2 von ElasticSearch (aktuelle Version ist 5.2) in Betrieb nehmen können.

Der Vorteil der Verwendung von Spring Data ElasticSearch als eine Implementierung von JPA ist das Spring Data eine High-Level API zur Verfügung stellt. Dies erlaubt es dem Entwickler den Zugriff auf Low-Level API von ElasticSearch zu vermeiden. Darüber hinaus wird das Mapping der Objektstruktur auf die ElasticSearch-Dokumente vom Framework komplett übernommen. Eine Alternativen zur Verwendung von Spring Ddata ElasticSearch wäre ein direkter Zugriff auf das ElasticSearch REST API oder die Verwendung von dem [ElasticSearch](#)

### Java Client.

Eine Weiterentwicklung der Persistenzarchitektur wäre das Einfügen einer Relationalen Datenbank. In diesem Fall könnte man nur die Informationen zum ElasticSearch übertragen, über welche effizient gesucht werden sollen. Eine Synchronisierung zwischen einer relationalen Datenbank und ElasticSearch könnte zum Beispiel durch den Einsatz von [Logstash](#) oder durch die Definition von einem [Batch-Prozess nach OASP](#) eingerichtet werden.

**Figure 1 Weiterentwicklung der Persistenzarchitektur**



## 7.3 Benutzungsoberfläche

### 7.3.1 Überblick

Die Benutzeroberfläche der Buchsuche basiert auf einer standard OASP4J Projektstruktur und nutzt das AngularJS Webframework. Ein OASP4JS Projekt kann über [Yeoman](#) generiert werden, somit steht die Grundstruktur des Frontends fest. Das Projekt kann für weitere Funktionen mit ng-modules erweitert werden.

Nozama powered by OASP4JS [Meine Bücher](#) [Logout](#)

Suche

Suche

Suche

Buchtitel

Autor

Buchtitel und Autor

Suchergebnis

Titel	Autor	Verlag	Datum	Bewertung	Herkunft
A Passage to India	Albertha Block	Chambers Harrap	1913	★★★★★ ★★★★★	nozama
Jacob Have I Loved	Albertha Wiza	Canongate Books	1728	★★★★★ ★★★★★	nozama
Vanity Fair	Albertha Quitzon	Verso Books	1877	★★★★★ ★★★★★	nozama
Endless Night	Albertha O'Keefe	Bella Books	1809	★★★★★ ★★★★★	nozama
The Lathe of Heaven	Albertha Sauer	Shoemaker & Hoard Publishers	1779	★★★★★ ★★★★★	nozama
Stranger in a Strange Land	Albertha Beier	Borgo Press	1711	★★★★★ ★★★★★	nozama
The Parliament of Man	Albertha Lesch	BBC Books	1933	★★★★★ ★★★★★	nozama
Behold the Man	Albertha Goodwin	Harper & Brothers	1888	★★★★★ ★★★★★	nozama
His Dark Materials	Albertha Langworth	Indiana University Press	1855	★★★★★ ★★★★★	nozama
The Needle's Eye	Albertha Armstrong	Shambhala Publications	1791	★★★★★ ★★★★★	nozama

## 7.3.2 GUI Elemente

### 7.3.2.1 Menüzeile

Nozama powered by OASP4JS [Meine Bücher](#) [Logout](#)

#### 7.3.2.1.1 Meine Bücher

Der Menüpunkt leitet auf den MS Meine Bücher weiter. Von dort kann über das Menü wieder auf die Buchsuche gewechselt werden.

URL: /nozama/#/

#### 7.3.2.1.2 Logout/Login

Je nachdem ob ein Benutzer ein- oder ausgeloggt ist steht die Schaltfläche für den Login oder Logout bereit.

Bei einem Login wird auf den MS Mein Bücher weitergeleitet

URL: /nozama/#/login?url=<AusgangsURL>

##### 7.3.2.1.2.1 Parameter

- AusgangsURL: Hier kann die URL der aktuellen Seite übergeben werden, um nach einem Login wieder auf diese zurück geleitet zu werden. Im aktuellen System ist diese aber auf die URL der Buchsuche festgelegt. Nach dem Login wird bei der Weiterleitung auf die Buchsuche die userId als Parameter in der ULR mitgeliefert.

Die Nutzerbehandlung im Frontend basiert komplett auf der userId-Variable. Wird ein Nutzer ausgeloggt, wird diese Variable auf null gesetzt. Es ist somit nur eine rudimentäre Nutzerverwaltung vorhanden, und bietet auch keine Sicherheitsaspekte wie Passwortverwaltung, verschlüsselte Übermittlung oder ähnliches.

### 7.3.2.2 Suchbereich

## Suche

### Suche

- Buchtitel**
- Autor**
- Buchtitel und Autor**

#### 7.3.2.2.1 Suchfeld

Das Suchfeld dient zur Eingabe des Suchbegriffs, das kann ein Titel oder ein Autor sein.

Get Request: /v1/book/search?q=<Suchbegriff>&searchBy=<Suchoption>

##### 7.3.2.2.1.1 Parameter

- q: Übergebener Suchbegriff
- searchBy (optional): Übergabe der Suchoption, wenn nicht vorhanden wird nach Buchtitel und Autor gesucht

Als Rückgabe der Sucher erhält das Frontend ein Array mit 20 Objekten (Büchern) {id, title, author, publisher, publishedAt}

#### 7.3.2.2.2 Suchoptionen

Es kann ausgewählt werden, wonach mit dem eingegebenen Suchbegriff gesucht werden soll. Durch Auswahl von Titel oder Autor kann gezielt gesucht werden, bei Auswahl der dritten Option wird nach beiden Feldern gesucht.

## 7.3.2.3 Suchergebnis

## Suchergebnis

Titel	Autor	Verlag	Datum	Bewertung	Herkunft
A Passage to India	Albertha Block	Chambers Harrap	1913	 ★★★★★  ★★★★★	
Jacob Have I Loved	Albertha Wiza	Canongate Books	1728	 ★★★★★  ★★★★★	
Vanity Fair	Albertha Quitzon	Verso Books	1877	 ★★★★★  ★★★★★	
Endless Night	Albertha O'Keefe	Bella Books	1809	 ★★★★★  ★★★★★	
The Lathe of Heaven	Albertha Sauer	Shoemaker & Hoard Publishers	1779	 ★★★★★  ★★★★★	
Stranger in a Strange Land	Albertha Beier	Borgo Press	1711	 ★★★★★  ★★★★★	
The Parliament of Man	Albertha Lesch	BBC Books	1933	 ★★★★★  ★★★★★	
Behold the Man	Albertha Goodwin	Harper & Brothers	1888	 ★★★★★  ★★★★★	
His Dark Materials	Albertha Langworth	Indiana University Press	1855	 ★★★★★  ★★★★★	
The Needle's Eye	Albertha Armstrong	Shambhala Publications	1791	 ★★★★★  ★★★★★	

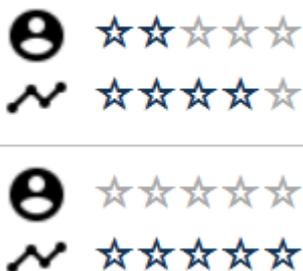
Das Ergebnisse einer Suche ist eine Liste mit genau 20 Büchern, sortiert nach Passgenauigkeit des Suchbegriffs.

Aufgrund der gegebenen Anforderungen werden folgende Felder angezeigt:

- **Titel:** Titel des Buches
- **Autor:** Autor des Buches
- **Verlag:** Verlag des Buches
- **Datum:** Erscheinungsjahr des Buches
- **Bewertung:** Obere Reihe zur Eingabe der eigenen Bewertung, untere Reihe zur Anzeige der Durchschnittsbewertung eines Buches
- **Herkunft:** Herkunft des Buches, nozama für die eigene Datenbank, amazon für den Import aus der Amazon Datenbank

## 7.3.2.4 Bewertung

## Bewertung



#### 7.3.2.4.1 Eigene Bewertung

Die obere Zeile dient der Bewertung eines Buches durch den Nutzer, diese Zeile ist nur dann vorhanden, wenn ein Nutzer eingeloggt ist. Durch das Auswählen einer Bewertung (Anzahl Sterne) wird diese zusammen mit der BuchID und der UserID an den MS Meine Bücher übermittelt. {id, rating, userId}

Die verwendete Funktion im Controller: rateFunction()

Der Post Request, der bei einer Bewertung aufgerufen wird: /v1/books/{bookId}/rate/{rating} + {userId=x}

##### 7.3.2.4.1.1 Parameter

- bookId: String mit der BuchID
- rating: Die Bewertung des Nutzers für ein Buch
- userId: Die NutzerID des aktuell eingeloggten Nutzers

#### 7.3.2.4.2 Durchschnittsbewertung

Die untere Zeile zeigt die durchschnittliche Bewertung eines Buches an. Die Durchschnittsbewertung wird bei einer Suchanfrage abgerufen, hierfür wird ein Array mit allen 20 Büchern {id} an die API vom MS Meine Bücher gesendet, als Rückgabe bekommt das Frontend ein Array mit der Durchschnittsbewertung jedes Buches {id, average}

Die verwendete Funktion im Controller: getRating()

Der Get Request, der beim Abrufen der Durchschnittsbewertungen aufgerufen wird: /v1/books/ratings/{id1,...,id20}

##### 7.3.2.4.2.1 Parameter

- id1,...,id20: Die Liste mit den 20 BuchIDs des Suchergebnisses

#### 7.3.2.5 Herkunft



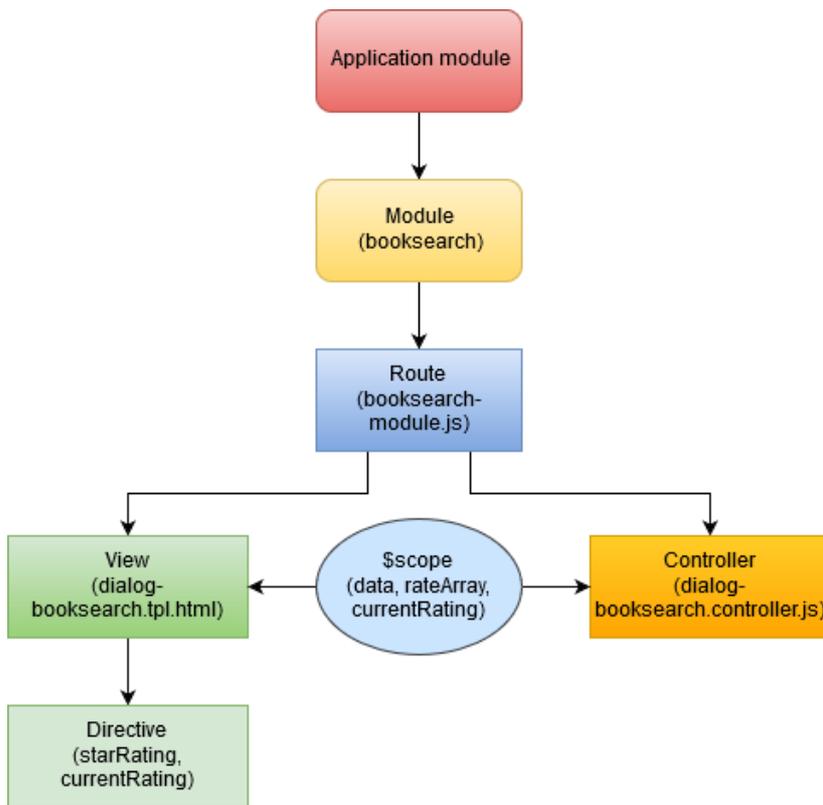
Diese beiden Symbole stellen die Herkunft eines Buches im Suchergebnis dar. Das linke Logo zeichnet ein Buch aus, das in der systemeigenen Datenbank vorhanden ist. Das rechte Logo weist darauf hin, dass ein Buch über die Amazon API in die eigene Datenbank importiert wurde.

### 7.3.3 Booksearch Architektur

Das folgende Diagramm zeigt eine grobe Architektur des Frontends der Buchsuche nach AngularJS.

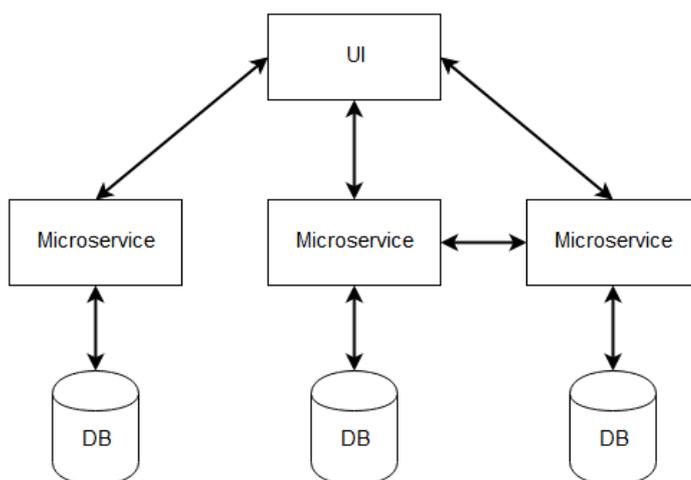
Der Grundbaustein der Anwendung bilden die Module, für die Buchsuche wurde das Modul "booksearch" hinzugefügt. Dieses Modul beinhaltet die gesamte Logik der Buchsuche im OASP4JS Projekt. Das Modul vereint alle weiteren Komponenten der Anwendung die eine AngularJS Anwendung ausmachen. Die Route (booksearch.module.js) ist für das Routing innerhalb der Anwendung zuständig und sorgt dafür, dass beim Aufruf des Frontends auf die Seite der Buchsuche (View) weitergeleitet wird und die entsprechenden Controller (dialog-booksearch.controller.js) bereitgestellt werden. Die Controller behandeln die Daten, die für die Funktionalität der Seite benötigt werden, im Falle einer Suche, werden alle Ergebnisse der Suche in einem Array (\$scope.data[]) gespeichert und an die View übergeben. Für die Behandlung der Buchratings werden unter anderem die Variablen \$scope.rateArray[], welche die Liste der Durchschnittsbewertung enthält und \$scope.currentRating, die die aktuelle Bewertung eines Buches beinhaltet, benötigt. Diese \$scope-Variablen vermitteln die Daten zwischen den Views und den Controllern. Für die Bewertungsfunktion werden zusätzliche Directives (starRating, currentRating) verwendet. Diese Directives werden verwendet, um die

Anzeige der Buchbewertungen dynamisch anzupassen und diese nach den Werten der \$scope-Objekten zu verändern.



## 7.4 Verteilung

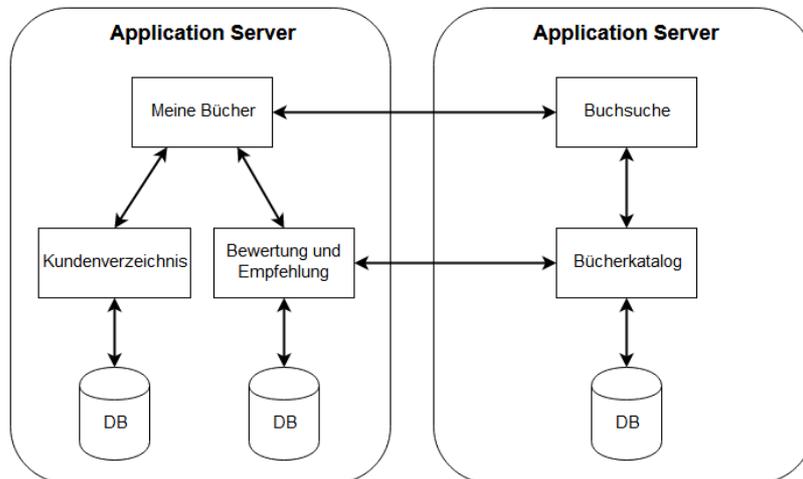
### 7.4.1 Microservice-Architektur



Die Microservice-Architektur sieht eine Verteilung sowohl in der horizontalen als auch in der vertikalen Achse vor. Auf der vertikalen Achse erfolgt die Verteilung in Form von Clients für die Interaktion mit dem Benutzer, der Anwendungsschicht, die die einzelnen Microservices und somit die Geschäftsaktivitäten der Anwendung bereitstellt und der Datenschicht, die der Persistierung der Daten dient.

Die horizontale Verteilung bezieht sich auf die Aufteilung der Geschäftsaktivitäten in Form von Microservices. Diese bilden von einander getrennte Services und sollten lediglich über genau definierte Schnittstellen (APIs) und geeignete Kommunikationsformen (REST) kommunizieren und somit lediglich eine lose Kopplung aufweisen. (Microservices, IoT, and Azure; Bob Familiar; Apress; 2015)

### 7.4.2 Verteilung der Client- und Serverkomponenten im Projekt



Im Projekt erfolgte die Verteilung, wie oben beschrieben, sowohl horizontal als auch vertikal. Horizontal erfolgte die Verteilung der Geschäftsaktivitäten des Online-Bücherkatalogs in Form von drei Microservices Kundenverzeichnis, Bewertung und Empfehlungssystem sowie des Bücherkatalogs. Diese kommunizieren, wie in Abschnitt [Externe Schnittstellen](#) genauer beschrieben, über REST-APIs.

Vertikal erfolgte eine Aufteilung in Client- und Serverkomponenten für die Buchsuche als auch für das Bewertungs- und Empfehlungssystem, welche auf zwei Application Servern unabhängig voneinander deployed wurden.

## 7.5 Konfigurierbarkeit

Für die Konfiguration der Anwendung wird Spring Boot eingesetzt. Dadurch unterstützt OASP sowohl eine Annotation- (auch *Java based*) als auch XML-basierte Konfiguration. Zusätzlich ist es auch möglich [Spring Boot Starter dependency descriptors](#) zu benutzen. Durch das Verwenden von `@AutoConfiguration` kann Spring Boot nach Komponenten suchen und diese automatisch konfigurieren.

Die erste Möglichkeit Spring Data Elasticsearch zu konfigurieren wäre die notwendigen Konfigurationen für Elasticsearch Cluster und Entitäten-Mapping in JSON-Dateien (esbook-mappings.json und elasticsearch-settings.json) abzulegen und mittels `@org.springframework.data.elasticsearch.annotations.Setting` und `@org.springframework.data.elasticsearch.annotations.Mapping` Annotationen in POJO-Klassen zu übernehmen. Dieses hat aber in unserem Fall nicht funktioniert. Die Ursache dafür konnte nicht gefunden werden.

Deswegen wurde eine andere Möglichkeit genutzt, nämlich die Konfigurierung über Java [Beans](#). Dazu wurde eine Klasse namens `ElasticSearchConfiguration` erstellt. Die Klasse selbst wurde mit `@Configuration` Annotation beschrieben. Die Methoden wurden mit `@Bean` Annotation signiert. Da das Spring Data Elasticsearch unter der Haube [ElasticSearch Java Client](#) für die Kommunikation mit Elasticsearch Cluster benutzt, überschreiben wir in unserem

Konfigurations-Bean die Instanziierung von dem Client und lesen die bereits erwähnte elasticsearch-settings.json Konfigurationsdatei direkt aus. Das bedeutet, dass die Einstellungen für ElasticSearch Cluster jedes Mal beim Systemstart ausgelesen werden.

Die Mapping-Einstellungen sollen aber dagegen für jedes DAO-Objekt explizit definiert werden. Deswegen wird komplett auf die JSON-Datei basierte Konfigurierung verzichtet. Stattdessen wird jede Klasseeigenschaft explizit bei der Verwendung von

**@org.springframework.data.elasticsearch.annotations.Id**

und

**@org.springframework.data.elasticsearch.annotations.Field(type, store, index, indexAnalyzer, searchAnalyzer)**

Spring Data ElasticSearch Annotationen beschrieben.

## 7.6 Codegenerierung

### 7.6.1 Projektstruktur OASP4J

Die Generierung eines Standard OASP4J Projektes gelingt über das Tool Maven.

Befehl: `mvn -DarchetypeVersion=2.2.0 -DarchetypeGroupId=io.oasp.java.templates -DarchetypeArtifactId=oasp4j-template-server archetype:generate -DgroupId=io.oasp.application -DartifactId=sampleapp -Dversion=0.1-SNAPSHOT -Dpackage=io.oasp.application.sampleapp`

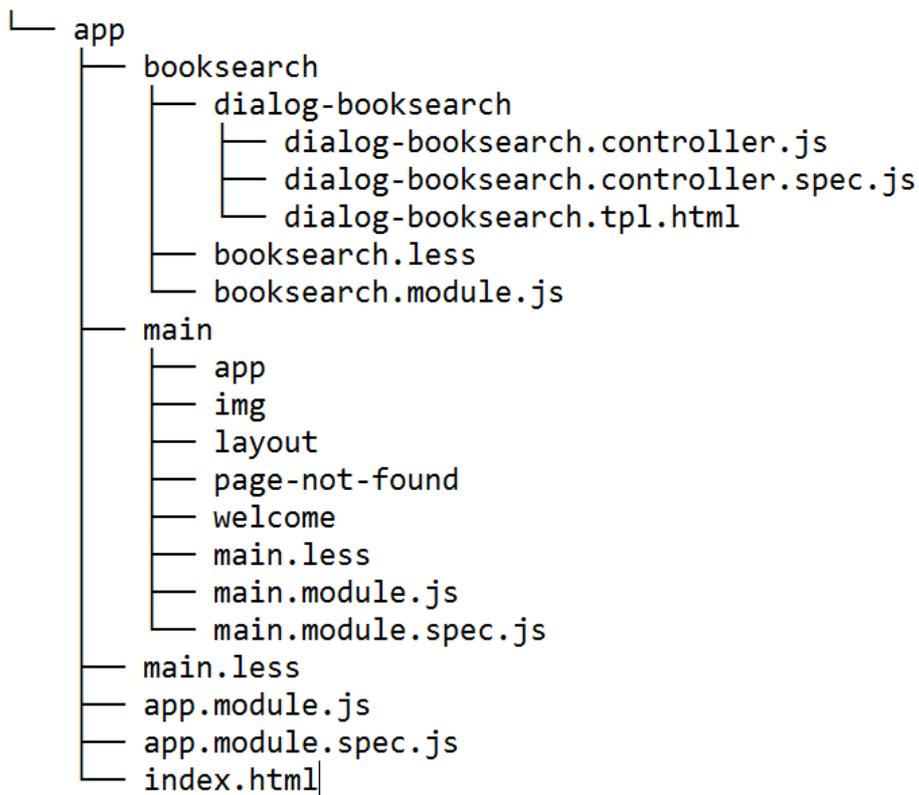
Über diesen Befehl wird ein OASP4J Templateprojekt in der Version 2.2.0 generiert.

Eine Übersicht über die Komponenten und Layer dieses MS kann in Kapitel [Bausteinsicht](#) gefunden werden.

Der Codegenerator für ein OASP4J Projekt erzeugt kein leeres Grundgerüst für ein Backend, das man verwenden kann. Es beinhaltet Überbleibsel von der Beispielanwendung "Restaurant" von OASP. Hat man sich ein OASP4J Projekt generiert müssen zuerst alle Rückstände der Beispielanwendung entfernt werden um ein sauberes Grundgerüst zu erhalten.

### 7.6.2 Projektstruktur OASP4JS

Die folgende Abbildung zeigt die aktuelle Projektstruktur des Frondends für die Buchsuche.

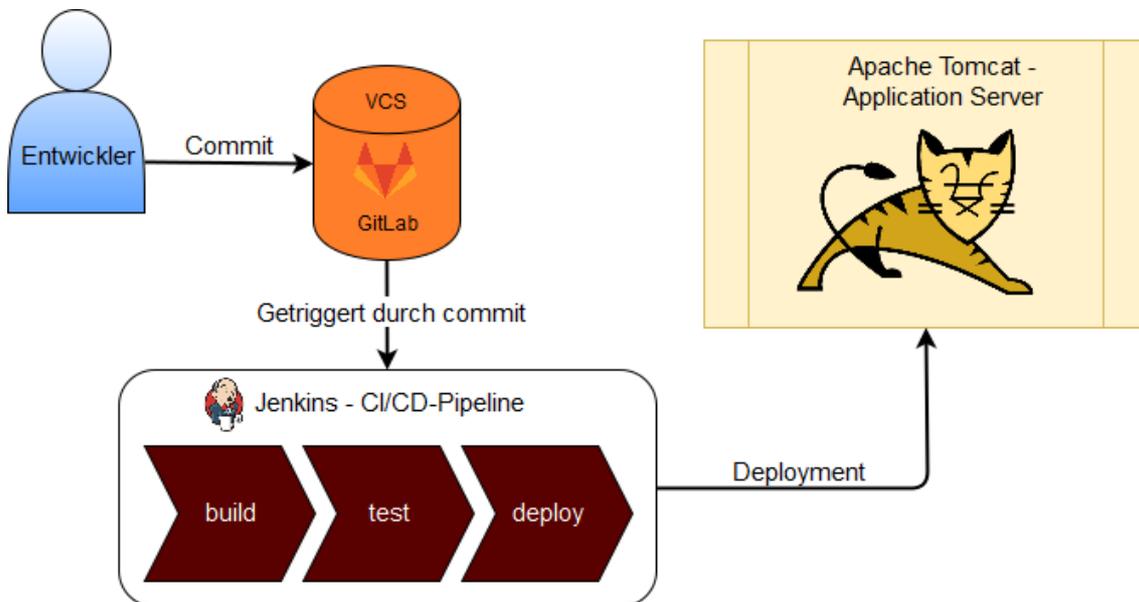
**Figure 2 Buchsuche - Projektstruktur**

Ein Standard Projekt in OASP4JS kann mittels dem Tool Yeoman generiert werden. Yeoman ist ein Plugin für npm und dient als Generator für verschiedene Projektstrukturen. Für ein OASP4JS Projekt kann mit dem generator-oasp-Generator ein Startprojekt generiert werden.

Die Struktur eines OASP4JS Frontends baut im wesentlichen auf Businessmodulen auf. Das heißt für jede Funktionalität wird ein eigenes Businessmodule hinzugefügt. Für die Buchsuche wurde nur ein Businessmodule verwendet, das die gesamte Businesslogic implementiert. Des weiteren befindet sich das Mainmodule im Projektbaum, das standardmäßig in einem OASP4JS Projekt vorhanden ist.

## 7.7 Buildmanagement

### 7.7.1 Abstrakter Ablauf der CI/CD-Pipeline



### 7.7.2 Versionsverwaltungssystem (VCS)

Als Versionsverwaltungssystem, zur Verwaltung der Repositories, wurde GitLab, welches auf Git basiert, verwendet. Da sowohl das Frontend als auch Backend als eigenständige Services entwickelt und bereitgestellt werden sollten wurden zwei einzelne Repositories angelegt. Das Repository des Frontends (<https://fsygs15.gm.fh-koeln.de:8888/GPAWS1617TEAM1/book-catalogue-frontend>) enthält dabei den Sourcecode der AngularJS-Application sowie die zugehörigen Build-Skripte und Konfigurationsdateien von Maven, Gulp und Bower. Das Repository des Backends (<https://fsygs15.gm.fh-koeln.de:8888/GPAWS1617TEAM1/book-catalogue-ms>) beinhaltet den Sourcecode der Serverkomponente und das Build-Skript von Maven sowie darüber hinaus die Konfigurationsdateien für die Checkstyleanalyse und der Anbindung von Elastic Search (siehe [Konfigurierbarkeit](#)).

Bei dem Masterbranch der jeweiligen Repositories handelt es sich um den Branch in Production. Dieser beinhaltet somit immer den aktuellen, fehlerfreien Entwicklungsstand der jeweiligen Services und wird durch das Build-Management-Tool Jenkins zu Beginn der CI/CD-Pipeline ausgecheckt. Zudem kamen im Projektverlauf Feature Branches zum Einsatz, um neue Funktionen hinzuzufügen, ohne diese sofort in Production zu übernehmen. Die Anwendung von Feature Branches, obwohl auf Continuous Integration gesetzt wurde, ist der Tatsache geschuldet, dass keine Testumgebung, die dem Umfang und Konfiguration der vorhandenen Server-Infrastruktur entsprach, zur Verfügung stand.

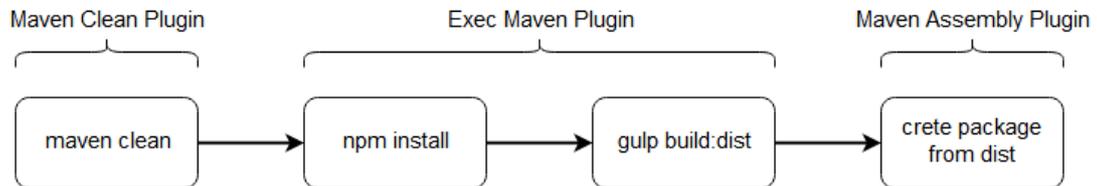
### 7.7.3 Build-Management-Tool

Als Build-Management-Tool kam Jenkins (<https://fsygs15.gm.fh-koeln.de/jenkins/>) in der Version 2.7.4 zum Einsatz. Jenkins bildet dabei das Grundgerüst der im Projekt zum Einsatz gekommenen CI/CD-Pipeline. Der Build-, Test-Vorgang wird dabei nach einem Commit auf den Masterbranch eines Repositories angestoßen und nach erfolgreichem Durchlauf erfolgt das Deployment auf den Application Server.

### 7.7.4 Build des Frontends als eigenständigen Service

Die nachfolgende Abbildung zeigt die Schritte des Buildvorgangs und die dabei verwendeten Maven-Plugins des OASP4JS-Clients. Damit die AngularJS-Application, welche das Frontend

der Buchsuche bildet, als eigenständiger Service auf dem Application Server deployed werden konnte, wurde ein MAVENSKRIPT erstellt, welches die dazu erforderlichen Schritte ausführt. Zunächst werden die bestehenden Ordner, Module und Dependencies mit Hilfe des Maven Clean Plugin gelöscht. Anschließend werden, durch das Exec Maven Plugin, die notwendigen Module installiert, die Dependencies aufgelöst und die AngularJS-Applikation kompiliert. Abschließend wird die kompilierte Applikation durch das Maven Assembly Plugin packetiert und als war.-File abgespeichert. Das zuvor erzeugte war.-File kann daraufhin auf dem Application Server unabhängig von der Backendkomponente deployed werden.



## 8 Risiken

### 8.1 1. Risiko: Aufwand der Implementierung

**Beschreibung:** Die Verwendung von OASP kann einerseits den Weg zum ersten Prototypen wesentlich verkürzen, kann aber andererseits zu den Schwierigkeiten führen, dass die OASP-Community nicht sehr groß und nicht in der Lage ist die Module des Frameworks stets aktuell zu halten. Dieses kann unter Umständen dazu führen, dass man auf aktuelle Technologien bzw. aktuelle Versionen von Dritt-Systemen verzichten muss bzw. diese lediglich mit großem Aufwand manuell einbinden kann.

**Eintrittswahrscheinlichkeit:** Hoch

**Schadenshöhe:** Mittel

**Massnahmen zur Risikominimierung:** OASP-Community beitreten und an der Weiterentwicklung des Frameworks teilnehmen / auf OASP verzichten

### 8.2 2. Risiko: Niedrige Softwarequalität durch das Fehlen von Change- und Qualitätsmanagement

**Beschreibung:** Es wurde nur eine Laufzeitumgebung (Production) für die Entwicklung bereitgestellt, die auch für das Team nicht direkt zugreifbar war. Dieses hat dazu geführt, dass das Team:

- in produktiver Umgebung getestet hat, was einem Benutzer der Dienste Probleme verursachen konnte.
- längere Wartezeiten für die Aktualisierungen von der Serverkonfigurationen in Kauf nehmen musste.

**Eintrittswahrscheinlichkeit:** Hoch

**Schadenshöhe:** Hoch

**Massnahmen zur Risikominimierung:** Es soll zumindest noch eine Testumgebung dem Team zu Verfügung gestellt werden. Darüber hinaus soll das Team einen direkten Zugriff auf den Test-Server haben.

## 9 Glossar

Begriff	Definiton
<b>API</b>	Das Application Programming Interface (API) ist eine Schnittstelle zur Anwendungsprogrammierung, die für die Abindung anderer Softwaresysteme zur Verfügung gestellt wird.
<b>Continuous Deployment</b>	Continuous Deployment (CD) ist ein Ansatz aus der Softwareentwicklung bei dem Methoden und Prozesse von agilen Vorgehensweisen mit der Produktion (IT-Betrieb) verknüpft werden. Somit besteht die Möglichkeit Software in einer hohen Geschwindigkeit iterativ zu entwickeln und bereits während der Entwicklung einsatzfähige Software zu liefern. Zu diesem Zweck werden hierbei möglichst alle Prozessschritte von der Entwicklung (Build und Tests) bis hin zum Deployment hochgradig automatisiert.
<b>Continuous Integration</b>	Continuous Integration (CI) beschreibt einen Prozess im Rahmen der Softwareentwicklung, bei dem Änderungen am Projekt kontinuierlich in das Gesamtsystem integriert werden. Die veränderten Komponenten werden hierbei mit den restlichen Bestandteilen zu einem lauffähigen Gesamtsystem zusammengefügt. Zur Unterstützung des Continuous Integration-Prozesses werden für die Verwaltung des Source Codes Version Control Systeme und für die Automatisierung der Build- und Testvorgänge entsprechende Werkzeuge eingesetzt.
<b>DevOps</b>	DevOps (Development und IT Operations) kann als Mentalität oder Kultur beschrieben werden und dient zur vollständigen Automatisierung des Entwicklungsprozesses unter Zuhilfenahme von Technologien und Werkzeugen. Setzt eine starke Kommunikation und Bindung zwischen der Entwicklung und dem Betrieb voraus. Bedarf bei der Einführung eine Neustrukturierung der vorhanden Unternehmensstrukturen und die Bereitschaft neue Methoden, Prozessen, Plattformen und Werkzeugen zu erlernen.
<b>JSON</b>	Die JavaScript Object Notation (JSON) stellt ein kompaktes, leicht zu lesendes Datenformat dar, welches unabhängig von der verwendeten Programmiersprache zum Datenaustausch dient.
<b>OASP</b>	Die, von Capgemini entwickelte, Open Application Standard Plattform (OASP) soll eine Lösung bieten, Anwendungen aufzubauen, die sowohl Best-in-Class-Frameworks und Bibliotheken, als auch in Industrie bewährte Praktiken und Code-Konventionen kombinieren.
<b>OASP4J</b>	Bei OASP4J handelt es sich um die Java-basierte Server-Komponente von OASP.
<b>OASP4JS</b>	Bei OASP4JS handelt es sich um Client-Komponente von OASP, welche auf AngularJS basiert.
<b>REST</b>	Representational State Transfer (REST) bezeichnet ein Programmierparadigmar für verteilte Systeme, bei dem Standards für die Kommunikation von Webservices über Schnittstellen definiert und genutzt werden.
<b>JIRA</b>	JIRA ist eine Webanwendung, die das Projektmanagement unterstützt. Jira stellt Funktionen zur Verwaltung von Anforderungsmanagement und Ablauforganisation bereit.

## 10 Verteilungsmatrix

Kapitel	Sebastian Domke	Ilya Sukhov	Nikolai Warkentin
Einführung und Ziele	0%	0%	100%
Randbedingungen	0%	100%	0%
Kontextabgrenzung	100%	0%	0%
Lösungsstrategie	0%	0%	100%
Bausteinsicht	0%	100%	0%
Laufzeitsicht	100%	0%	0%
Fachliche Strukturen und Konzepte	0%	100%	0%
Persistenz	0%	100%	0%
Benutzeroberfläche	0%	0%	100%
Verteilung	100%	0%	0%
Konfigurierbarkeit	0%	100%	0%
Codegenerierung	0%	0%	100%
Buildmanagement	100%	0%	0%
Risiken	0%	100%	0%
Glossar	100%	0%	0%
Scrum im Hochschulkontext	0%	0%	100%
Arbeit im Team	0%	0%	100%
Evaluierung von OASP im Microservicekontext	0%	100%	0%
Probleme/Lösungen im Projektverlauf	100%	0%	0%

## 10.1 Verteilungsmatrix Team 1 - Excel Sheet



GPA\_WS1617\_Ver...ix\_Team\_1.xlsx

# 11 Scrum im Hochschulkontext

Dieses Kapitel soll einen Überblick darüber gehen, wie das Scrumframework an den Hochschulkontext angepasst wurde und wie die Arbeit nach dieser Organisation funktioniert hat.

## 11.1 Die Teams

An diesem Projekt haben sich 6 Personen beteiligt, die in zwei separate Teams aufgeteilt wurden. Da es sich um ein Projekt im Microservicekontext handelte, war es geplant mindestens zwei Teams zu bilden, damit jedes Team an den entsprechenden Microservices arbeiten kann. Somit hatte jedes der Teams drei Mitglieder, wovon jedes eine Scrumrolle übernommen hat. Für ein solches Projekt war die Teamgröße mit drei Mitgliedern ein Minimum und hätte nicht kleiner ausfallen dürfen. Auf der anderen Seite war das Projekt mit nur drei Microservices und entsprechend nur zwei Teams sehr überschaubar, wodurch die Herausforderungen einer realen Microservicearchitektur nicht wirklich vorhanden waren. Die Herausforderungen, die es bei der Kommunikation zwischen den Teams geben kann, sind zum Teil auch in diesem Projekt entstanden, jedoch wäre dieser Aspekt bei weiteren Teams wesentlich schwieriger gewesen.

Für ein Projekt dieser Art wäre es interessant gewesen, wie das Projekt mit mehr Microservices und entsprechend mehr Projektteams verlaufen wäre und wie gut die Gruppenarbeiten mit mehr Teammitgliedern funktioniert hätten. Eine Planung von so einem Projekt ist aber natürlich von der Anzahl der Teilnehmer abhängig und kann nicht beeinflusst werden.

## 11.2 Scrumframework

### 11.2.1 Scrumteams

Da das Projekt auf dem Scrumframework basiert, waren auch die Scrumrollen im Team verteilt. Die drei Scrumrollen Scrum Master (Nikolai Warkentin), Product Owner (Sebastian Domke) und Team Member (Ilya Sukhow) wurden dabei auf die drei Teammitglieder verteilt, somit hat jedes Mitglied eine der drei Rollen vertreten. Die Teams hatten demnach die minimale Größe, um jede Scrum Rolle mindestens einmal besetzen zu können. In einem normalen Scrumteam wären idealerweise mehr Personen dabei, sodass es mehr Team Member geben würde.

Zusätzlich zu der geringen Teamgröße, waren die Rollen von Scrum Master und Product Owner doppelt belegt, denn alle Teammitglieder haben als normale Team Member im Projekt mitgewirkt und zusätzlich ihre zugewiesene Rolle übernommen. Idealerweise wäre die Rolle des Product Owners von einer Person außerhalb des Projektteams übernommen worden. So hätte sich eine externe Person um die Items im Product Backlog gekümmert und den Sprint mit dem Team zusammen geplant, was auch eher dem Vorgehen eines Scrum Projektes entspricht.

### 11.2.2 Scrum

#### 11.2.2.1 Organisation

Das Projekt war so weit es möglich war nach Scrum organisiert. Es gab insgesamt 4 Sprints in denen an dem Produkt gearbeitet wurde und dementsprechend auch 4 Sprint Review Meetings. Ein Sprint war genau 4 Wochen lang (ein Monat) und es wurde jede Woche an einem Arbeitstag (8 Stunden) zusammen im Team gearbeitet. Aufgrund der begrenzten Zeit, die die einzelnen Teammitglieder innerhalb eines Semesters für ein Projekt investieren konnten, war nicht mehr als ein Tag in der Woche möglich, um kollokiert im Team arbeiten zu können. In einem zukünftigen Projekt könnte versucht werden, die Projekte blockweise stattfinden zu lassen, sodass ein gewisser zusammenhängender Zeitraum für ein einziges Projekt genutzt wird und somit auch mehrere Tage in der Woche für dieses Projekt investiert werden könnten.

#### 11.2.2.2 Scrummeetings

In einem normalen Scrumprojekt gibt es nach jedem abgeschlossenen Sprint ein Sprint Review Meeting, eine Sprint Retrospective und ein Sprint Planning Meeting, zusätzlich werden jeden Arbeitstag Standup Meetings abgehalten, bevor mit der Arbeit begonnen wird. In diesem Projekt waren diese Meetings ebenfalls vorhanden. Das Sprint Review, Sprint Planning und Sprint Retrospective Meeting wurden am letzten Arbeitstag eines jeden Sprints abgehalten, wodurch ein halber Arbeitstag verloren ging und pro Sprint nur 3 1/2 Arbeitstage übrig blieben. Zusätzlich zu diesen Treffen, wurden alle zwei Wochen Scrum of Scrums Meetings abgehalten, in denen sich alle Teams zusammengefunden haben. Diese haben jeweils am ersten und am vorletzten Arbeitstag stattgefunden. Diese Treffen hätten stattdessen in der Mitte eines Sprints abgehalten werden sollen, da so etwas Zeit nach dem Sprint Planning vergangen wäre und Themen gesammelt wurden, die besprochen werden müssten und noch genügend Zeit geblieben wäre, um noch etwas bis zum kommenden Sprint Review Meeting vorbereiten zu können.

Die Standup Meetings wurden im Projektverlauf regelmäßig abgehalten, ergaben jedoch nicht den angestrebten Nutzen, da diese Meetings im Normalfall täglich stattfinden und nicht wie in diesem Projekt nur ein mal die Woche. So wurde diese Zeit verwendet sich daran zu erinnern, was diese Person beim letzten Mal gemacht hat und womit sie sich dieses Mal beschäftigen will.

Die Problematik für das Team bei den Sprint Review Meetings war, wie die Präsentation aufgebaut werden sollte. Zum einen standen die Lehrenden aus der Sicht des Guided Projekts im Fokus und zum anderen die Kunden des zu entwickelnden Produktes. Eine klare Rollenveteilung der Stakeholder zu Beginn des Projektes wäre für die Entwicklerteams von Vorteil gewesen.

## 12 Arbeit im Team

### 12.1 Erfolge

Da das Projektziel sich aus mehreren Unterzielen zusammensetzt, soll der Zielerreichungsgrad auch aus verschiedenen Blickwinkeln betrachtet werden. Zum einen ist es dem Team gut gelungen alle formalen (auch optionalen) Anforderungen an das Endprodukt nach dem 4. Sprint abzudecken. Damit hat das Team erfolgreich das Projektziel aus Stakeholdersicht von Professor Bente erreicht. Andererseits hat das Team an dem Teamprozess nach Scrum ständig gearbeitet, um die von Professor Bente als Lehrender vorgegebene Ziel nahe zukommen. Im ersten Schritt war es wichtig die drei Scrumrollen optimal zu besetzen. Obwohl diese zum Anfang des Projektes per Zufall zugewiesen worden sind, hat sich im weiteren Verlauf des Projektes gezeigt, dass diesbezüglich keine weiteren Anpassungen notwendig waren.

Neben der optimalen Rollenverteilung hat das Team sich vorgenommen an dem Teamprozess ständig zu arbeiten, um die Verbesserungspotentiale des Teams auszuschöpfen. Dies wurde durch eine sorgfältige Ausführung und Dokumentation aller in Scrum beschriebenen Aktivitäten erreicht. Die Erwartungen der Teammitglieder wurden nur halbwegs erfüllt. Einerseits haben die Gruppenmitglieder wertvolle praktische Erfahrungen mit Scrum gemacht und ein lauffähiges Produkt entwickelt und präsentiert. Andererseits waren die Einblicke in das Microservices-Thematik sehr limitiert und oberflächlich. Nach dem Abschluss des Projektes waren sich die Teammitglieder einig, dass es effizienter und effektiver wäre den Studenten innerhalb des Guided Projects eine Möglichkeit zu geben einzelne neue Microservices zu entwickeln und in eine bereits existierende Microservice-Umgebung einzubinden. Damit hätten die Studierenden näher an einer produktiven Lösung gearbeitet und mehr Erfahrungen im Umgang mit komplexen Systemen sammeln können.

Das Verbesserungspotential des Teams innerhalb des Scrum-Prozesses bei gegebenen Rahmenbedingungen ist ausgeschöpft worden. Weitere Hürden auf dem Weg zur Teamleistungssteigerung sind dagegen eher der Organisation des Guided Projektes zuzuschreiben. Zum einen ist die niedrige Anzahl der Mitglieder pro Team problematisch, da sie keine feinere Analyse des Teamprozesses zugelassen hat. Andererseits hat die breit formulierte Aufgabenstellung das Team dazu gebracht etwa ein Drittel seiner Zeit für die Aufgaben aufzuwenden, die nicht wirklich viel mit der eigentlichen Microservice-Entwicklung zu tun haben.

### 12.2 Probleme / Konflikte

Innerhalb des Teams kam es im Laufe des Projekts zu keinen größeren Konflikten. Dies ist zum einen darauf zurückzuführen, dass sich die Ansprüche und Erwartungen, der Teammitglieder von Team 1, an das Projekt auf einem annähernd gleichen Level bewegten. Dazu zählen unter anderem die individuellen Ziele, die zu Beginn des Projekts von den Teammitgliedern definiert wurden. Darin standen neben der praktischen Erfahrungen mit Scrum, dem Wissensausbau im Bereich von Microservices und den damit verbunden Technologien, vor allem die Verbesserung der Teamkompetenz und das erfolgreiche Abschließen des Moduls im Vordergrund.

Im Laufe des Team Process-Workshops zeigte sich, dass die Teammitglieder von Team 1 ein in etwa gleich hohes Maß an Motivation und Arbeitsbereitschaft mitbrachten, was zu einem angenehmen Teamklima beigetragen hat und bei einer starken Diskrepanz zu Konflikten hätte führen können. Ein weiterer wichtiger Punkt, der vermutlich dazu beigetragen hat, dass es im Team zu keinen großen Konflikten gekommen ist, ist die Durchführung von Analyse der eigenen Persönlichkeit und Charaktereigenschaften in Form von Selbst- und Fremdeinschätzung im Rahmen der Workshops. Dadurch war es den Teammitgliedern möglich ihre Stärken und auch ihre Schwächen aufzudecken und mit Hilfe des Feedbacks der anderen Teammitglieder gezielt an diesen zu arbeiten. Außerdem konnte unter Zuhilfenahme der vermittelten Feedbackregeln konstruktiv und sachlich über persönliche Dinge diskutiert werden,

wie es in vielen anderen Projekten in dieser Art und Weise ansonsten nicht möglich gewesen wäre beziehungsweise diese Dinge erst gar nicht thematisiert worden wären.

Probleme, die die Kommunikation und Absprache mit dem anderen Team als auch technische oder organisatorische Aspekte des Projekts betreffen, werden im Abschnitt [Probleme/Lösungen im Projektverlauf](#) genauer betrachtet.

## 13 Evaluierung von OASP im Microservicekontext

OASP, wie jedes anderes Framework, bringt sowohl zahlreiche Vorteile mit sich, die das Leben eines Entwicklers einfacher machen können, als auch einige Nachteile, die unter Umständen die Geschwindigkeit der Entwicklung etwas vermindern können.

Zu den klaren Vorteilen zählen:

- Eine klar definierte Architektur. Technische Architektur von OASP ist definiert durch [5 Schichten](#): Client, Service, Batch, Logic und Data Access. Zusätzlich gibt es eine Reihe von Querschnittskomponenten wie Security, Logging, Monitoring, Transaction- und Exception-Handling, Internationalization und Dependency Injektion. Darüber hinaus liefert die OASP-Dokumentation zahlreiche Informationen wie diese Blaupause im Code umzusetzen ist.
- Im Prinzip geht es bei OASP-Anwendungen um Spring Boot-basierten Maven-Anwendungen, was eine Integration in eine Spring Cloud Umgebung (EventBus, API-Gateway, Service Discovery, External Config, Monitoring, Load Balancer und andere Dienste) erleichtert.
- OASP ist, dank klar definierter Module, Kodierkonventionen und dem Einsatz von Spring Boot für die Konfiguration der Anwendung und für eine schnelle Prototyperstellung (sobald die Umgebung eingerichtet) gut geeignet.
- Dank modularem Aufbau weisen die OASP-Anwendungskomponenten ein hohes Grad an Testierbarkeit und Wiederverwendbarkeit auf.

Zu den wenigen Nachteilen kann man zuordnen:

- Die OASP-IDE, obwohl diese als ein Teil des **Open** Application Standard Platform definiert worden ist, ist diese nicht frei verfügbar bzw. muss manuell kompiliert werden:

*"If you are not member of Capgemini: We cannot distribute the package. Please consult [oasp4j-ide](#) to setup and configure the IDE manually. If you need help, please get in touch."*

Somit wird die Einstiegsbarriere um einiges höher gesetzt.

- Darüber hinaus stehen OASP-Projektschablonen nur für die OASP-IDE zur Verfügung. Leider bietet die OASP4J-Community kein Werkzeug für das Projekt-Bootstrapping an (wie z.B. [Spring Initializr](#)).
- Um ein neues Projekt anzulegen muss man entweder dieses über Kommandozeile generieren oder das Beispielpjekt von OASP nehmen und es an seine eigenen Bedürfnisse anpassen.
- Es wird auch kein Starter-Projekt bei GitHub angeboten wie es bei anderen open source Projekten üblich ist (z.B. [Angular2 Starter](#)).

Spring Framework ist mehr marktorientiert und strebt nicht nach Einhaltung von aktuellen Standards (z.B. Spring 5 ist JavaEE 6 basiert). JavaEE wiederum ist aus einer Reihe von Standards gewachsen, was die Verbreitung von der Plattform etwas verlangsamt. Mit anderen Worten steht Spring eher für Innovation, wobei JavaEE mehr nach Standardisierung strebt. OASP macht den Eindruck von einer Lösung, die einen Spagat zwischen die beiden Welten versucht. Einerseits wird in der Dokumentation mehrmals angedeutet, dass die einzelnen Komponenten nach Standards aufgebaut sind, andererseits besteht das Framework zum Teil aus veralteten Spring-Komponenten. Nicht zu vergessen ist, dass hinter JavaEE ein Java Community Process steht und hinter Spring die VMware und open source Community (durch welche z.B. zahlreiche [Spring Data](#) module entwickelt worden sind). Hinter OASP steht nur Capgemini mit wenigen Entwicklern, was z.B. zu Verzögerungen bei der Aktualisierung von unterstützten Komponenten (z.B. Spring Boot) oder mangelnder Dokumentation ([Guide Monitoring](#)) führt.

Deswegen stellt sich die Frage, welche entscheidenden Vorteile OASP im Vergleich zu JavaEE und, vor allem, da OASP stark von Spring-Komponenten abhängig ist, zu Spring hat? Die Antwort auf diese interessante Frage konnte, leider, im Laufe des Projektes nicht gefunden werden.

## 14 Probleme/Lösungen im Projektverlauf

Der nachfolgende Abschnitt befasst sich mit Problemen, die im Laufe des Projekts aufgetreten sind. Dabei wird zwischen Problemen technischer und organisatorischer Natur unterschieden.

### 14.1 Technische Probleme

Die technischen Probleme im Projekt waren vorrangig OASP spezifisch. Dabei kam es bereits zu Beginn des Projekts zu Schwierigkeiten ein frisch erzeugtes OASP4J-Projekt (siehe [Codegenerierung](#)) auf dem Application Server zu deployen. Die Probleme bestanden dabei zum Einen darin, dass der festgelegte LogPath bei einem Tomcat-Server, wie er bei diesem Projekt als Application Server eingesetzt wurde, in einer Produktivumgebung standardmäßig keine Schreibrechte besitzt. Somit kam es beim Deployment zu einem Logback configuration error, welcher jedoch schnell durch eine Änderung des LogPath in der logback.xml behoben werden konnte. Der passende Commit, der den Fix des Problems enthält, kann hier <https://fsygs15.gm.fh-koeln.de:8888/GPAWS1617TEAM1/book-catalogue-ms/commit/cd2dade0e9fd659555af646012cb652ab546a656#diff-0> eingesehen werden. Zusätzlich wurde ein Issue (<https://github.com/oasp/oasp4j/issues/512>) im Repository von OASP4J erzeugt, um auf das Problem aufmerksam zu machen.

Ein weiteres Problem bestand ebenfalls direkt nach dem Erzeugen eines neuen OASP4J-Projekts in der Version 2.1.1. Hierbei kam es zu einem Konflikt zwischen zwei konkurrierenden Log-Bibliotheken. Die Lösung des Problems bestand darin, dass die Dependency der slf4j-Bibliothek aus der cor/pom.xml entfernt wurde und somit ausschließlich log4j zum Einsatz kam. Der dazu passenden Commit kann hier <https://fsygs15.gm.fh-koeln.de:8888/GPAWS1617TEAM1/book-catalogue-ms/commit/9a92b0fea729505e74ae4a3d58808ba8d6dc6894> eingesehen werden. Ebenfalls wurde zu diesem Problem ein Issue-Report im OASP4J-Repository erzeugt (<https://github.com/oasp/oasp4j/issues/513>).

Durch den Einsatz von OASP als Grundlage für die Umsetzung der Anforderungen kam es zu weiteren technischen Problemen.

Die strikten Vorgaben der Java-Version (JRE 1.7.x) und Maven-Version (maven 3.2.5) bei OASP in der Version 2.1.1 erschwerte zum einen die Einrichtung einer funktionsfähigen Entwicklungsumgebung der Teammitglieder. Des Weiteren stellte sich im Projektverlauf heraus, dass ein Update der Spring-Boot-Version über die Version 1.3.3 bei OASP nicht möglich war. Somit kam es beispielsweise bei der Umsetzung der unscharfen Suche mittels Elastic Search zu Problemen, da auf Grund der veralteten Spring-Boot-Version auf eine stark veraltete Elastic Search-Version (1.7.2) zurückgegriffen werden musste, die nur über eine geringe Dokumentation und einen eingeschränkten Funktionsumfang verfügt. Weitere Details zu dieser Problematik können dem Abschnitt [Persistenz](#) entnommen werden.

### 14.2 Organisatorische Probleme

Zusätzlich zu den technischen Problemen traten im Projektverlauf organisatorische Problemen auf.

Im Rahmen dieser Probleme kam es unter anderem zu einem Konflikt zwischen den beiden Scrumteams. Dieser bestand darin, dass unser Team wenige Tage vor dem Review von Sprint 3 eine Änderung an einer API unseres Microservice vorgenommen hat, der sich auch auf die Funktionsweise des Empfehlungs- und Bewertungssystem von Team 2 ausgewirkt hat. Die Änderung bestand im Detail darin, dass die IDs der Bücher des Bücherkatalogs zuvor als Integer definiert waren und anschließend in Form von Strings. Die Änderung ist auf den Wechsel von der relationalen Datenbank zu Elastic Search zurückzuführen und diente dazu die Anforderung einer unscharfen Suche innerhalb unseres Bücherkatalogs zu realisieren und das von uns definierte Sprintziel zu erreichen.

Um einen solchen Konflikt zu verhindern, hätten die Teams zu Beginn des Projekts definieren müssen, wie mit Änderungen, die beispielsweise die APIs betreffen, umgegangen wird. Hierzu

hätte beispielsweise ein aktives Change Management betrieben werden können, welches die API Specificationen der Teams einschließt und genau festlegt wie und zu welchem Zeitpunkt Änderungen kommuniziert und durchgeführt werden müssen.

Ein weiteres Problem stellte die starke Abhängigkeit im Bereich DevOps dar. Durch die vorgegebene Nutzung der vorhandenen Server-Infrastruktur für die CI/CD-Pipeline und den Application Server, um eine vergleichbare Projektsituation wie sie bei Capgemini zum Einsatz kommt zu erzeugen, kam es, trotz hervorragender Unterstützung und Arbeit von Marco Reitano und Alex Maier, zu Verzögerungen im Arbeitslauf. Dies bezieht sich vor allem auf die Fehlerfindung bei Problemen im Bereich der CI/CD-Pipeline und dem Application Server sowie bei Änderungswünschen der eingesetzten Technologien, wie beispielsweise Versionsänderungen des Elastic Search-Plugins, auf dem Application Server.

Eine mögliche Lösung für dieses Problem, wäre beispielsweise die Auslagerung der CI/CD-Pipeline und des Application Servers in die Cloud. Somit wäre es möglich, dass sich die Teams eigenständig um die Erstellung und Konfiguration kümmern und diese nach ihren Bedürfnissen einrichten können. Darüber hinaus wäre in diesem Zusammenhang die Verwendung von Docker sinnvoll, um neben der Produktivumgebung eine Testumgebung aufbauen zu können. Somit könnte die Testumgebung mit einfachen Mitteln auch lokal betrieben werden und auf diese Weise wäre es dem Team möglich Änderungen zunächst innerhalb einer einheitlichen Testumgebung zu testen bevor diese in der Produktivumgebung deployed werden.

Ein zusätzliches organisatorisches Problem bestand, aus unserer Sicht, in der Vorgabe der Dokumentationsform nach arc42 als Grundlage für den geforderten Evaluationsbericht. Dabei empfanden wir es als schwierig die Anforderungen des Evaluationsberichts mit der Gliederung und der Dokumentationsform einer Architekturdokumentation nach arc42 zu verknüpfen. Es wäre wünschenswert gewesen, wenn entweder auf die Vorgabe den Evaluationsbericht nach arc42 zu strukturieren verzichtet worden wäre oder eine detailliertere Erläuterung des genauen Anspruchs an den geforderten Berichts stattgefunden hätte.