# Einführung und Ziele

Anforderungen:

- Anlegen von Medien (Bücher und CDs)

- Nutzerauthentifizierung

- Ausleihen von Medien

- zurückgeben von Medien

Architekturziele:

- Skalierbar

- Anfangs auf commodity-Hardware lauffähig

Stakeholder:

- Prof. Boettcher

Beschreibt die wesentliche Anforderungen und treibenden Kräfte, die Softwarearchitekten und Entwicklungsteams berücksichtigen müssen. Dazu gehören die

* zugrunde liegenden Geschäftsziele, wesentliche Aufgabenstellung und essenzielle fachliche Anforderungen an das System
* Qualitätsziele für die Architektur
* relevante Stakeholder und deren Erwartungshaltung

## Aufgabenstellung

**Inhalt.**

Es soll eine webapp erstellt werden mit deren Hilfe eine unbestimmte Anzahl an Nutzern ihre Bücher/CDs einstellen können um diese an andere Nutzer zu verleihen. Jeder Nutzer kann sich selbst einen User erstellen, welcher aber von einem Administrator aktiviert werden muss um Missbrauch zu verhindern.

Aufgeteilt wurde das Projekt in mehrere Sprints:

**Shareit v1:**

* Definieren einer einfachen REST-API
* Sie definieren eine eigene Vorgehensweise für die Behandlung von Fehlern in dieser REST-API
* Vorgehensweise für die Behandlung von Fehlern in der REST-API definieren
* Implementierung und Testen einer einfach REST-API
* deployen der Webapp bei hakuo

**Shareit v2:**

* Registrieren eines neuen Benutzers
* Aktivieren eines Benutzers
* Anmelden eines bereits registrierten Benutzers am System
* Abmelden eines angemeldeten Benutzers vom System

**Shareit v3:**

* Beseitigung aller Bugs
* Persistenzierung aller benoetigten Objekte.

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

## Qualitätsziele

|  |  |
| --- | --- |
| **Ziel** | **Metrik** |
| Performance | 500ms Response-time bei 50 aktiven Usern |
| Code Coverage | 80% |
| Funktionalität | Alle geforderten Funktionalitäten müssen vorhanden sein |

## Stakeholder

|  |  |  |
| --- | --- | --- |
| Rolle | Kontakt | Erwartungshaltung |
| *Stakeholder* | *Prof. Axel Boettcher* | *Funktionales System mit automatischen Tests.* |

# Randbedingungen

Entwickelt wird das System in der Programmiersprache Java.

Aufteilung des Systems in mehrere Schichten:

- Datenzugriffsschicht

- Geschäftslogikschicht

- API-Schicht

- Darstellungsschicht

Gebaut und deployed werden soll das Projekt mithilfe von Maven.

Die webapp soll mithilfe von Maven gebaut und auf heroku deployed werden.

# Kontextabgrenzung

# Heroku?

**Inhalt.**

Die Kontextabgrenzung grenzt das System von allen Kommunikationspartnern (Nachbarsystemen und Benutzerrollen) ab. Sie legt damit die externen Schnittstellen fest.

Differenzieren Sie fachlichen Kontext (fachliche Ein- und Ausgaben) und technischen Kontext (Kanäle, Protokolle, Hardware), falls nötig.

**Motivation.**

Die fachlichen und technischen Schnittstellen zu Kommunikationspartnern gehören zu den kritischsten Aspekten eines Systems. Stellen Sie sicher, dass Sie diese komplett verstanden haben.

**Form.**

Verschiedene Optionen:

* Diverse Kontextdiagramme
* Listen von Kommunikationspartnern mit deren Schnittstellen

## Fachlicher Kontext

|  |  |  |
| --- | --- | --- |
| **Kommunikationspartner** | **Eingabe** | **Ausgabe** |
| Nutzer | GET-Request  PUT/PUSH-Request mit Json | Angeforderte Information,  invalid json,  sonstige Fehlermeldungen (z.B. Nutzer anlegen → Nutzername bereits vergeben) |
| Administrator | GET-Request  PUT/PUSH-Request mit Json | Erweiterte Ausgaben |
| Prof. Boettcher | Sämtlicher Code | Feedback |

Festlegung **aller** Kommunikationspartner (Nutzer, IT-Systeme, …) mit Erklärung der fachlichen Ein- und Ausgabedaten oder Schnittstellen. Zusätzlich bei Bedarf fachliche Datenformate oder Protokolle der Kommunikation mit den Nachbarsystemen.

**Motivation.**

Alle Beteiligten müssen verstehen, welche fachlichen Informationen mit der Umwelt ausgetauscht werden.

## Technischer Kontext

**Many stakeholders make architectural decision based on the technical interfaces between the system and its context. Especially infrastructure or hardware designers decide these technical interfaces.**

Technische Schnittstellen (Kanäle, Übertragungsmedien) zwischen dem System und seiner Umwelt. Zusätzlich eine Erklärung (*mapping*), welche fachlichen Ein- und Ausgaben über welche technischen Kanäle fließen.

**Motivation.**

Viele Stakeholder treffen Architekturentscheidungen auf Basis der technischen Schnittstellen des Systems zu seinem Kontext.

Insbesondere Infrastruktur- oder Hardwareentwickler entscheiden auch über diese technischen Schnittstellen.

**Form.**

Beispielsweise UML Deployment-Diagramme mit den Kanälen zu Nachbarsystemen, begleitet von einer Tabelle, die Kanäle auf Ein-/Ausgaben abbildet.

**<Diagramm oder Tabelle>**

**<optional: Erläuterung der externen technischen Schnittstellen>**

**<Mapping fachliche auf technische Schnittstellen>**

# Lösungsstrategie

Es wurde entschieden die Webapp mit Hilfe von Microservices umzusetzen.

Dies ermöglicht es uns einfach und schnell zu skalieren sowie das Projekt anfangs auf kleiner, schwacher Hardware umzusetzen.

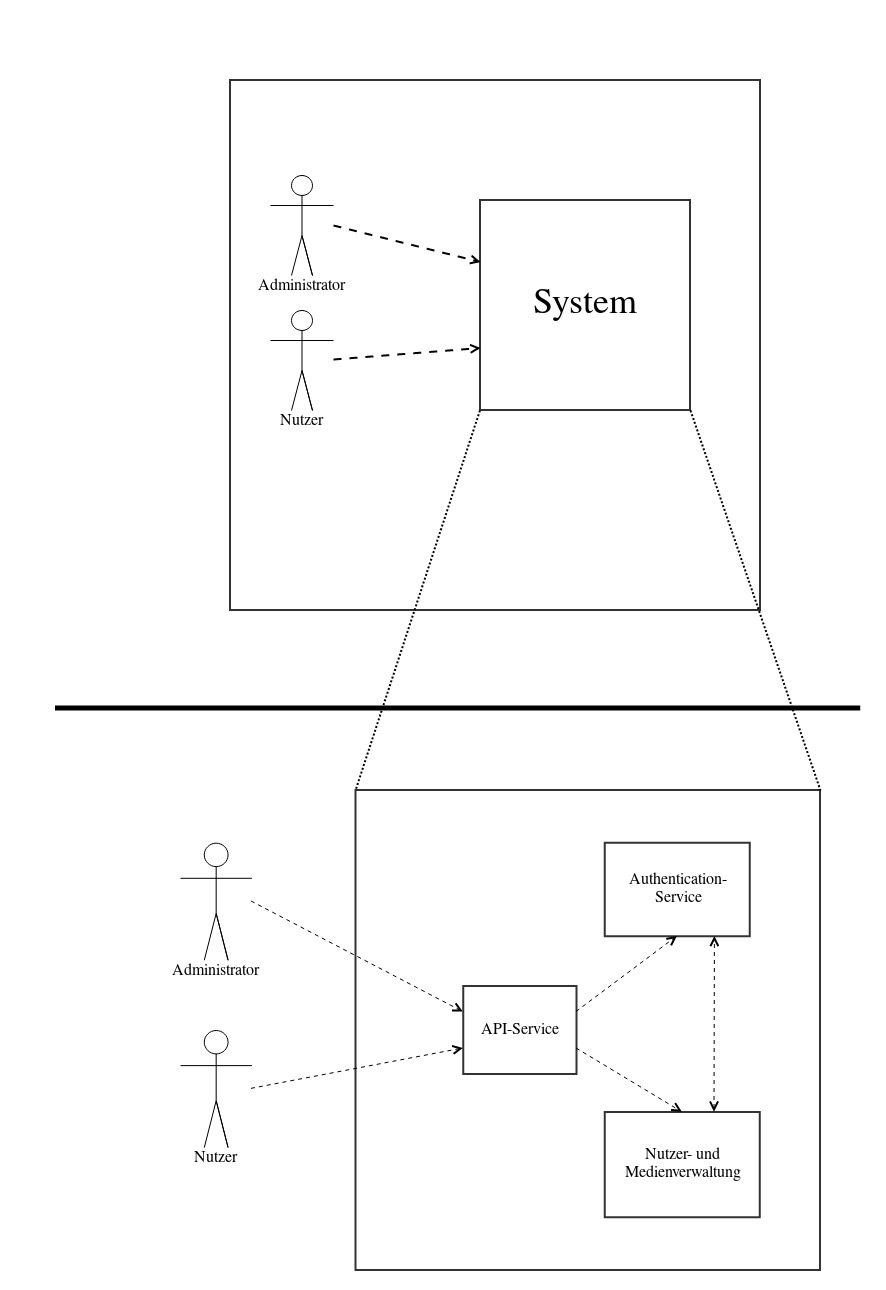
Um diese Trennung zu vollziehen wurde auf Dependenci-Injection welche uns zusätzlich das Testen mit Mock-Objekten ermöglicht und REST-Schnittstellen gesetzt.

Jeder Service (Authentifizierung, Medium-Verwaltung) muss potentiell ueber eine REST-Schnittstelle ansprechbar sein.

Um die gewünschte Qualität sicherzustellen beinhaltet die “definition of done” das schreiben von adäquaten tests und das dokumentieren im Code.

Eine gewisse Flexibilität bei der Umsetzung erreichen wir durch das entwickeln nach den SCRUM-Prinzipien.

# Bausteinsicht



|  |  |
| --- | --- |
| **Name** | **Verantwortung** |
| *API-Service* | *Bereitstellung aller externen Ressourcen.*  *Diese beinhalten z.B. anzeigen aller Buecher, login und logout eines Nutzer, anlegen eines Buches, ...* |
| *Authentication-Service* | *Authentifiziert einen Nutzer, erstellt einen Token, loescht den Token beim logout.* |
| Nutzer- und Medienverwaltung | Verwaltet und persistenziert die Daten aller Buecher/CDs und der Nutzer |

### API-Service:

### Dieser Service stellt dem Nutzer alle externen Ressourcen bereit.

### Er interagiert mit dem Authentication-Service um den Nutzer ein- und auszuloggen und seine Berechtigungen zu überprüfen.

### Bereitgestellte Ressourcen:

### -login

### -logout

### -getAllBooks

### -createBook

### -activateUser

### -createUser

### Nutzer- und Medienverwaltung:

### Verwaltet und Persistenzierung alle Nutzer- und Mediendaten.

### Authentication-Service:

### Der Authentication-Service verwaltet alle eingeloggten User und deren Tokens. Er interagiert mit der User- und Medienverwaltung um den User initial auf gültigen Namen, Passwort und Rolle zu überprüfen.

# Laufzeitsicht

Der User spricht bei all seinen Interaktionen mit dem API-Service.

Dieser leitet die Ressourcen login und logout an den Authentifizierungs-Service weiter, welche beim ersten Zugriff eines Nutzers dessen Daten von der Nutzer- und Medienverwaltung bezieht.

Versucht der User nun eine andere Ressource anszusprechen, welche eine Authentifizierung benötigt, überprüft der API-Service zuerst den Zustand des Nutzers (eingeloggt oder nicht) und dessen Berechtigungen auf diese Ressource.

Ist beides gegeben so wird die Anfrage aufbereitet an die Nutzer- und Medienverwaltung geschickt um z.B. ein neues Medium anzulegen oder eine Liste aller Medien auszugeben.

Die Nutzer- und Medienverwaltung fuehrt die entsprechende Aktion aus, persistenziert die Daten und reicht das Ergebniss an den API-Service weiter, welcher wiederrum den Nutzer informiert.

# Verteilungssicht

# Sämtliche Services werden auf Heroku gehostet.

# Über den Standort oder sonstige Hardwarespezifische Informationen kann keine Auskunft erteilt werden.

# Querschnittliche Konzepte

Zur persistenzierung benutzen wir 5 Daten-Modelle:

- User (enthält einen Nutzer mit seinem Nutzername, Passwort, Rollen und sonstigen Informationen)

- Medium – ein Medium mit einem Titel und einer Beschreibung (Abstrakte Klasse)

- Book – ein Buch mit seinen Eigenschaften (z.B. ISBN extends Medium)

- Disc – eine CD/DVD/Bluray mit ihren Eigenschaften (z.B. Barcode extends Medium)

- Copy – ein “physikalisches” Book/Disk welche einem Besitzer (Nutzer) zugeordnet ist.

Durch diese Aufteilung gehen uns keine Informationen ueber ein Medium verloren, selbst wenn keine Copy davon im System vorhanden ist.

Zusätzlich können Bücher/CDs im vornherein angelegt werden.

Mithilfe der Medium-Klasse kann eine bessere Suche implementiert werden. So können bei einer Suche nach “Herr der Ringe – Die Gefährten” sowohl Bücher als auch CDs angezeigt werden.

Um die Logik-Schicht flexibel zu halten wurden sämtliche benötigten Methoden in einem Interface (z.B. MediumDataAccess) festgehalten, welche das Mindestmaß an Funktionen definiert.

Jede Implementierung einer Klasse in der Log-Schicht muss sein dementsprechendes Interface implementieren.

Die letztendlich genutzte Klasse kann daher schnell und einfach ausgetauscht werden um verschiedene Zusatzfunktionen bereit zu stellen.

# Entwurfsentscheidungen

**Inhalt.**

Wichtige, teure, große oder riskante Architektur- oder Entwurfsentscheidungen inklusive der jeweiligen Begründungen. Mit "Entscheidungen" meinen wir hier die Auswahl einer von mehreren Alternativen unter vorgegebenen Kriterien.

Wägen Sie ab, inwiefern Sie Entscheidungen hier zentral beschreiben, oder wo eine lokale Beschreibung (z.B. in der Whitebox-Sicht von Bausteinen) sinnvoller ist. Vermeiden Sie Redundanz. Verweisen Sie evtl. auf Abschnitt 4, wo schon grundlegende strategische Entscheidungen beschrieben wurden.

**Motivation.**

Stakeholder des Systems sollten wichtige Entscheidungen verstehen und nachvollziehen können.

**Form.**

Verschiedene Möglichkeiten:

* Liste oder Tabelle, nach Wichtigkeit und Tragweite der Entscheidungen geordnet
* ausführlicher in Form einzelner Unterkapitel je Entscheidung
* ADR ([Architecture Decision Record](http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions)) für jede wichtige Entscheidung

# Qualitätsanforderungen

**Inhalt.**

1. Funktionalität
2. Modularisierung
3. Performance
4. Code-Coverage
5. Dokumentation

## Qualitätsszenarien

## 1. Funktionalität:

## Alle geforderten Funktionalitäten müssen korrekt implementiert werden.

## 2. Modularisierung:

## Für sämtliche Zugriffe auf die Logik- und Data-Schicht sollen Interfaces genutzt werden um einen einfachen Austausch der funktionalen Klassen zu ermöglichen.

## 3.Performance

## Bei 50 gleichzeitig aktiven Nutzern soll die Antwortzeit des Systems unter 500ms liegen.

## 4. Code Coverage

## Die Code-Coverage soll mithilfe von EclEmma gemessen mehr als 80% betragen.

## 5. Dokumentation

## Sämtlicher Code ist nach “javadoc” zu kommentieren.

# Risiken und technische Schulden

# 1. Ueberkomplizierung durch Modularisierung

# 2.

**Inhalt.**

Eine nach Prioritäten geordnete Liste der erkannten Architekturrisiken und/oder technischen Schulden.

**Motivation.**

"Risikomanagement ist Projektmanagement für Erwachsene" (Tim Lister, Atlantic Systems Guild.)

Unter diesem Motto sollten Sie Architekturrisiken und/oder technische Schulden gezielt ermitteln, bewerten und Ihren Management-Stakeholdern (z.B. Projektleitung, Product-Owner) transparent machen.

**Form.**

Liste oder Tabelle von Risiko und/oder technischen Schulden, eventuell mit vorgeschlagenen Maßnahmen zur Risikovermeidung, Risikominimierung oder dem Abbau der technischen Schulden.

# Glossar

**Inhalt.**

Die wesentlichen fachlichen und technischen Begriffe, die Stakeholder im Zusammenhang mit dem System verwenden.

Nutzen Sie das Glossar ebenfalls als Übersetzungsreferenz, falls Sie in mehrsprachigen Teams arbeiten.

**Motivation.**

Sie sollten relevante Begriffe klar definieren, so dass alle Beteiligten

1. diese Begriffe identisch verstehen, und
2. vermeiden, mehrere Begriffe für die gleiche Sache zu haben.

* Zweispaltige Tabelle mit <Begriff> und <Definition>
* Eventuell weitere Spalten mit Übersetzungen, falls notwendig.

|  |  |
| --- | --- |
| Begriff | Definition |
| *<Begriff-1>* | *<Definition-1>* |
| *<Begriff-2* | *<Definition-2>* |