 Share-It

2017-06-21

# Einführung und Ziele

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

* Verwaltung von Benutzerdaten
* Verwaltung von Exemplaren
* Ausleihe und Ruckgabe der Exemplare

Standardablauf:

1. User loggt sich ein
2. User legt ein Buch an
3. User schaut nach ob dieses Buch eingetragen ist
4. User loggt sich aus

## Aufgabenstellung

**Inhalt.**

Siehe [Anforderungsdokument]

## Qualitätsziele

**Inhalt.**

|  |  |
| --- | --- |
| Qualitätsziele | Szenarien |
| Robustheit | System sollte bei fehlerhafter Eingabe weiter funktionieren |
| Modulares System | Einfach skalier- und erweiterbar |
| Anlegen von Büchern | User soll Bücher anlegen können |
| Einsehen von Büchern | User sollen den Bücherbestand angezeigt bekommen können. |
| Authenztifizierung von Usern | Zugangskontrolle für User |

## Stakeholder

|  |  |  |  |
| --- | --- | --- | --- |
| Rolle | Beschreibung | Abnahmerelevanz | Erwartungen |
| Administratoren | Wartung des Systems | Hoch | Einfach zu warten |
| Studenten – User | Benutzer des Systems | Hoch | Einfache Bedienbarkeit |
| Hochschulleitung | Finanzierer des Systems | Mittel | Kostengünstig |
|  |  |  |  |

# Randbedingungen

**Lizenz**

GitHub repository.

Deployment auf Heroku. https://www.heroku.com/policy/tos

Nutzung von Hibernate (LGPL 2.1) und log42 (Apache License, Version 2.0).

**Technisch**

Modulares System.

Zielumgebung: Java auf Heroku.

Zeitliche Begrenzung bis zum 2.7.2017.

# Kontextabgrenzung

Standalone auf Heroku.

## Fachlicher Kontext

## Technischer Kontext

Deployment auf Heroku.

JSON Objekt Aufbau  
{  
 "name" : "bookName",  
 "author" : "name",  
 "ISBN" : number,  
}  
  
  
  
POST:  
Legt ein neues Buch an  
Respond: StatusCode:  
 - 200 alles ok buch wurde angelegt  
 - 400 es existiert bereits ein identisches buch  
 - 401 ungültige eingabe  
Respond JSON obj bei 400 und 401  
 {  
 "detail" : "Fehler beschreibung"  
 }  
  
Get:  
lieft information zu allen angelegten Büchern  
Resond Status Code:  
 - 200 alles ok  
 - 400 es gibt keine angelegten bücher  
Respond JSON obj bei 200  
 {  
 <Beliebige anzahl>  
 {  
 "name" : "bookName","author" : "name", "ISBN" : number,  
 }  
 <\Beliebige anzahl>

Aktuelle REST API im Repository.

# Lösungsstrategie

Nutzung der Microservices mit Schichtenarchitektur.

Datenhaltung mit Hibernate.

|  |  |
| --- | --- |
| Aufgabe/Problem | Lösungsansatz |
| Skalierfähigkeit/Modularität | Mircoservices |
| Robustheit | Prüfung der Eingabe |
| Schnittstelle | REST Schnittstelle mit JSON |

|  |  |  |
| --- | --- | --- |
| Qualitätsmerkmal | Szenario | Maßnahmen |
| Robustheit | Fehlerhafte Implementirerung | Junit Tests |
|  |  |  |

**Inhalt.**

Kurzer Überblick über die grundlegenden Entscheidungen und Lösungsansätze, die Entwurf und Implementierung des Systems prägen. Hierzu gehören:

* Technologieentscheidungen
* Entscheidungen über die Top-Level-Zerlegung des Systems, beispielsweise die Verwendung gesamthaft prägender Entwurfs- oder Architekturmuster
* Entscheidungen zur Erreichung der wichtigsten Qualitätsanforderungen
* relevante organisatorische Entscheidungen, beispielsweise für bestimmte Entwicklungsprozesse oder Delegation bestimmter Aufgaben an andere Stakeholder.

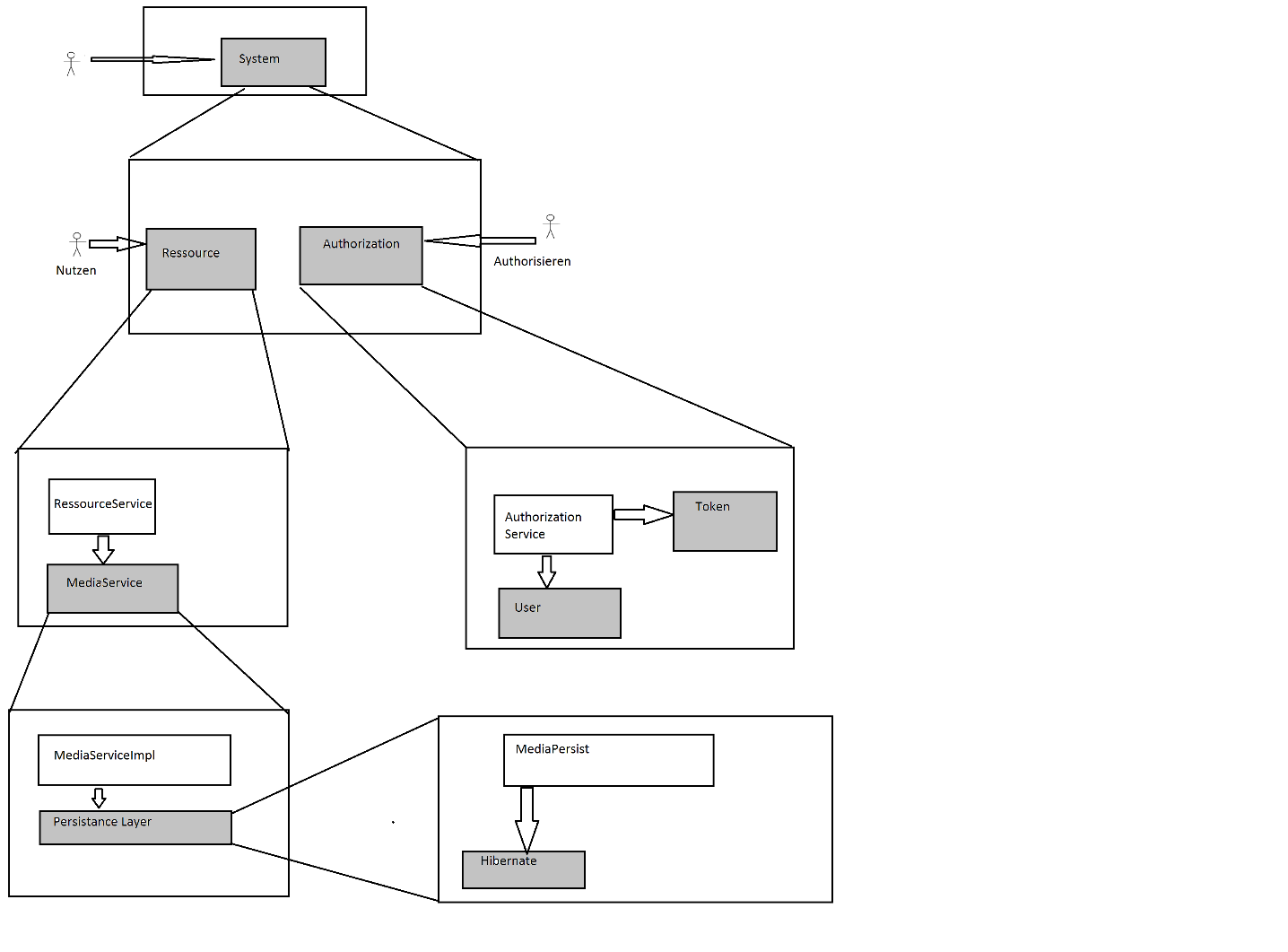
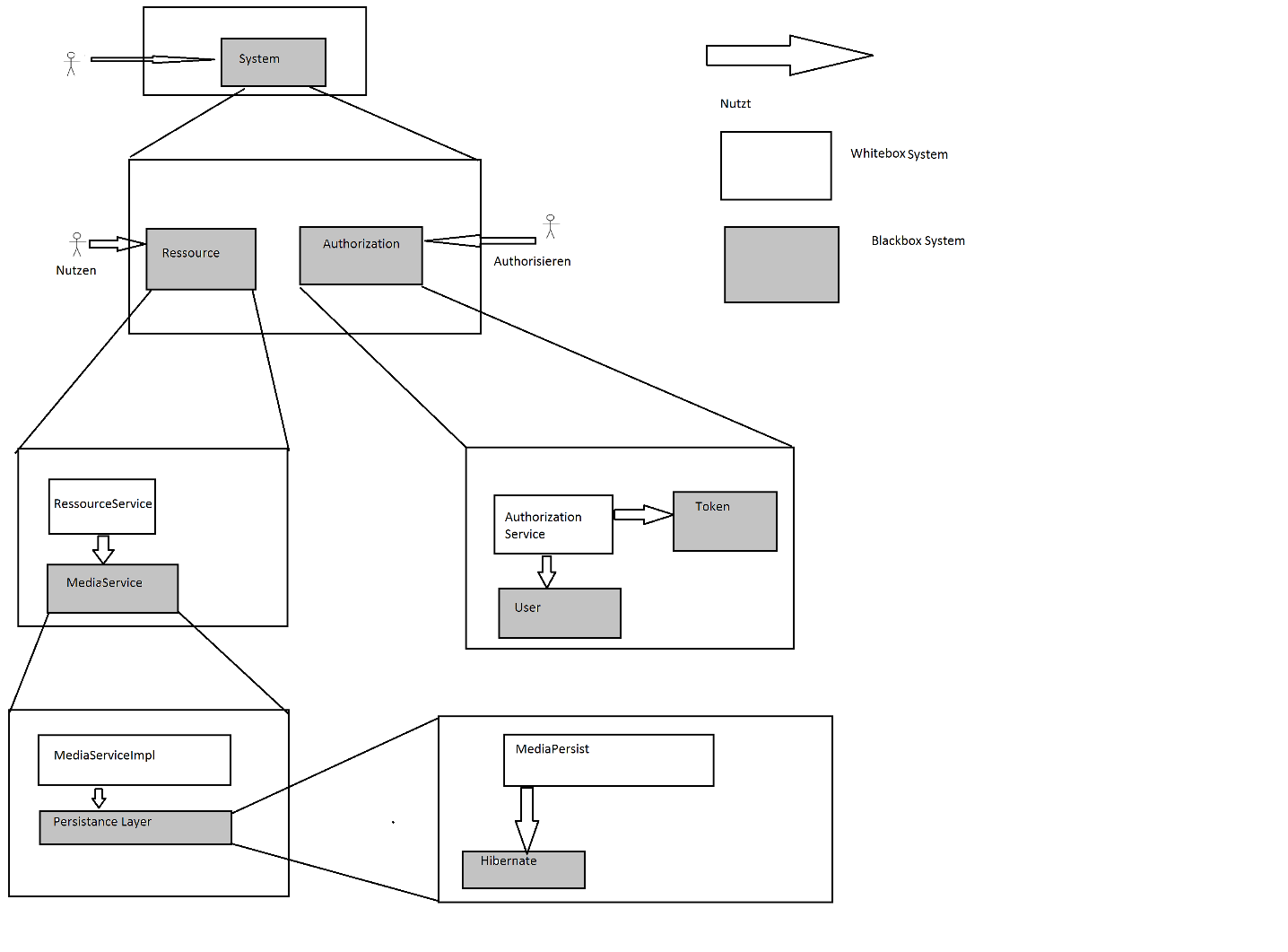
**Motivation.**

Diese allerwichtigsten Entscheidungen bilden wesentliche „Eckpfeiler“ der Architektur. Von ihnen hängen meistens viele weitere Entscheidungen oder Implementierungsregeln ab.

**Form.**

Fassen Sie die zentralen Entwurfsentscheidungen **kurz** zusammen. Motivieren Sie ausgehend von Aufgabenstellung, Qualitätszielen und Randbedingungen, was Sie entschieden haben und warum Sie so entschieden haben. Verweisen Sie eher auf weitere Ausführungen in Folgeabschnitten.

# Bausteinsicht



**Inhalt.**

Diese Sicht zeigt die statische Zerlegung des Systems in Bausteine (Module, Komponenten, Subsysteme, Klassen, Interfaces, Pakete, Bibliotheken, Frameworks, Schichten, Partitionen, Tiers, Funktionen, Makros, Operationen, Datenstrukturen…) sowie deren Beziehungen.

Diese Sicht sollte in jeder Architekturdokumentation vorhanden sein . In der Analogie zum Hausbau bildet die Bausteinsicht den *Grundrissplan*.

**Motivation.**

Behalten Sie den Überblick über den Quellcode, indem Sie die statische Struktur des Systems durch Abstraktion verständlich machen.

Damit ermöglichen Sie Kommunikation auf abstrakterer Ebene, ohne zu viele Implementierungsdetails offenlegen zu müssen.

**Form.**

Die Bausteinsicht ist eine hierarchische Sammlung von Blackboxen und Whiteboxen (siehe Abbildung unten) und deren Beschreibungen.



# Laufzeitsicht

**Inhalt.**

Diese Sicht erklärt konkrete Abläufe und Beziehungen zwischen Bausteinen in Form von Szenarien aus folgenden Bereichen:

* Wichtige Abläufe oder *Features*: Wie führen die Bausteine der Architektur die wichtigsten Abläufe durch?
* Interaktionen an kritischen externen Schnittstellen: Wie arbeiten Bausteine mit Nutzern und Nachbarsystemen zusammen?
* Betrieb und Administration: Inbetriebnahme, Start, Stop.
* Fehler- und Ausnahmeszenarien

Anmerkung: Kriterium für die Auswahl der möglichen Szenarien (d.h. Abläufe) des Systems ist deren Architekturrelevanz. Es geht nicht darum, möglichst viele Abläufe darzustellen, sondern eine angemessene Auswahl zu dokumentieren.

**Motivation.**

Sie sollten verstehen wie (Instanzen von) Bausteine(n) Ihres Systems ihre jeweiligen Aufgaben erfüllen und zur Laufzeit miteinander kommunizieren.

Nutzen Sie solche Szenarien in der Dokumentation hauptsächlich zur besseren Kommunikation mit Stakeholdern, die statische Modelle (z.B. Bausteinsicht, Verteilungssicht) weniger verständlich finden.

**Form.**

Für die Beschreibung von Szenarien gibt es zahlreiche Ausdrucksmöglichkeiten. Nutzen Sie beispielsweise:

* Nummerierte Schrittfolgen oder Aufzählungen in Umgangssprache
* Aktivitäts- oder Flussdiagramme
* Sequenzdiagramme
* BPMN oder EPKs (Ereignis-Prozessketten)
* Zustandsautomaten
* …

## *<Bezeichnung Laufzeitszenario 1>*

* <hier Laufzeitdiagramm oder Ablaufbeschreibung einfügen>
* <hier Besonderheiten bei dem Zusammenspiel der Bausteine in diesem Szenario erläutern>

## *<Bezeichnung Laufzeitszenario 2>*

…

## *<Bezeichnung Laufzeitszenario n>*

…

# Verteilungssicht

**Inhalt.**

Die Verteilungssicht beschreibt:

1. die technische Infrastruktur, auf der Ihr System ausgeführt wird, mit Infrastrukturelementen wie Standorte, Umgebungen, Rechnern, Prozessoren, Kanälen und Netztoplogien sowie sonstigen Bestandteilen und
2. die Abbildung von (Software-)Bausteinen auf diese Infrastruktur.

Häufig laufen Systeme in unterschiedlichen Umgebungen ab, beispielsweise Entwicklung-/Test- oder Produktionsumgebungen. In solchen Fällen sollten Sie alle relevanten Umgebungen aufzeigen.

Nutzen Sie die Verteilungssicht insbesondere, wenn Ihre Software auf mehr als einem Rechner, Prozessor, Server oder Container abläuft oder Sie Ihre Hardware sogar selbst konstruieren.

Aus Softwaresicht genügt es auf die Aspekte zu achten, die für die Softwareverteilung relevant sind. Hardwarearchitekten können bei Bedarf die Infrastruktur mit beliebigen Details beschreiben.

**Motivation.**

Software läuft nicht ohne Infrastruktur. Diese zugrundeliegende Infrastruktur beeinflusst Ihr System und/oder querschnittliche Lösungskonzepte, daher müssen Sie diese Infrastruktur kennen.

Das oberste Verteilungsdiagramm könnte bereits in Ihrem technischen Kontext enthalten sein, mit Ihrer Infrastruktur als EINE Black-Box. Jetzt zoomen Sie in diese Infrastruktur mit weiteren Verteilungsdiagrammen hinein:

* Die UML stellt mit Verteilungsdiagrammen (Deployment diagrams) eine Diagrammart zur Verfügung, um diese Sicht auszudrücken. Nutzen Sie diese, evtl. auch geschachtelt, wenn Ihre Verteilungsstruktur es verlangt.
* Falls Ihre Infrastruktur-Stakeholder andere Diagrammarten bevorzugen, die Prozessoren und Kanäle zeigen, sind die hier ebenfalls einsetzbar.

## Infrastruktur Ebene 1

An dieser Stelle beschreiben Sie (als Kombination von Diagrammen mit Tabellen oder Texten):

* die Verteilung des Gesamtsystems auf mehrere Standorte, Umgebungen, Rechner, Prozessoren oä. sowie die physischen Verbindungskanäle zwischen diesen,
* wichtige Begründungen für dieser Verteilungsstruktur,
* Qualitäts- und/oder Leistungsmerkmale dieser Infrastruktur,
* Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur

Für mehrere Umgebungen oder alternatives Deployment kopieren Sie diesen Teil von arc42 für alle wichtigen Umgebungen.

***<Übersichtsdiagramm>***

Begründung

*<Erläuternder Text>*

Qualitäts- und/oder Leistungsmerkmale

*<Erläuternder Text>*

Zuordnung von Bausteinen zu Infrastruktur

*<Beschreibung der Zuordnung>*

## Infrastruktur Ebene 2

An dieser Stelle können Sie den inneren Aufbau (einiger) Infrastrukturelemente aus Ebene 1 beschreiben.

Für jedes Infrastrukturelement kopieren Sie die Struktur aus Ebene 1.

### *<Infrastrukturelement 1>*

*<Diagramm + Erläuterungen>*

### *<Infrastrukturelement 2>*

*<Diagramm + Erläuterungen>*

…

### *<Infrastrukturelement n>*

*<Diagramm + Erläuterungen>*

# Querschnittliche Konzepte

**Inhalt.**

Dieser Abschnitt beschreibt übergreifende, prinzipielle Regelungen und Lösungsansätze, die an mehreren Stellen (=*querschittlich*) relevant sind.

Solche Konzepte betreffen oft mehrere Bausteine. Dazu können vielerlei Themen gehören, beispielsweise:

* fachliche Modelle,
* eingesetzte Architektur- oder Entwurfsmuster,
* Regeln für den konkreten Einsatz von Technologien,
* prinzipielle, meist technische, Festlegungen übergreifender Art,
* Implementierungsregeln

**Motivation.**

Konzepte bilden die Grundlage für *konzeptionelle Integrität* (Konsistenz, Homogenität) der Architektur und damit eine wesentliche Grundlage für die innere Qualität Ihrer Systeme.

Manche dieser Themen lassen sich nur schwer als Baustein in der Architektur unterbringen (z.B. das Thema "Sicherheit"). Hier ist der Platz im Template, wo Sie derartige Themen geschlossen behandeln können.

**Form.**

Kann vielfältig sein:

* Konzeptpapiere mit beliebiger Gliederung,
* übergreifende Modelle/Szenarien mit Notationen, die Sie auch in den Architektursichten nutzen,
* beispielhafte Implementierung für insbesondere technische Konzepte,
* Verweise auf "übliche" Nutzung von Standardframeworks (beispielsweise die Nutzung von Hibernate als Object/Relational Mapper.

**Struktur.**

Eine mögliche (nicht aber notwendige!) Untergliederung dieses Abschnittes könnte wie folgt aussehen (wobei die Zuordnung von Themen zu den Gruppen nicht immer eindeutig ist):

* Fachliche Konzepte
* User Experience (UX)
* Sicherheitskonzepte (Safety und Security)
* Architektur- und Entwurfsmuster
* Unter-der-Haube
* Entwicklungskonzepte
* Betriebskonzepte



## *<Konzept 1>*

*<Erklärung>*

## *<Konzept 2>*

*<Erklärung>*

…

## *<Konzept n>*

*<Erklärung>*

# 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.**

Dieser Abschnitt enthält möglichst alle Qualitätsanforderungen als Qualitätsbaum mit Szenarien. Die wichtigsten davon haben Sie bereits in Abschnitt 1.2 (Qualitätsziele) hervorgehoben.

Nehmen Sie hier auch Qualitätsanforderungen geringerer Priorität auf, deren Nichteinhaltung oder -erreichung geringe Risiken birgt.

**Motivation.**

Weil Qualitätsanforderungen die Architekturentscheidungen oft maßgeblich beeinflussen, sollten Sie die für Ihre Stakeholder relevanten Qualitätsanforderungen kennen, möglichst konkret und operationalisiert.

## Qualitätsbaum

**Inhalt.**

Der Qualitätsbaum ( a la ATAM) mit Qualitätsszenarien an den Blättern.

**Motivation.**

Die mit Prioritäten versehene Baumstruktur gibt Überblick über die oftmals zahlreichen Qualitätsanforderungen.

* Baumartige Verfeinerung des Begriffes „Qualität“, mit "Qualität" oder Nützlichkeit als Wurzel.
* Mindmap mit Q-Oberbegriffen als Hauptzweige

In jedem Fall sollten Sie hier Verweise auf die Szenarien des folgenden Abschnittes aufnehmen.

## Qualitätsszenarien

**Inhalt.**

Konkretisierung der (in der Praxis oftmals vagen oder impliziten) Qualitätsanforderungen durch (Qualitäts-)Szenarien.

Diese Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht.

Wesentlich für die meisten Softwarearchitekten sind zwei Arten von Szenarien:

* Nutzungsszenarien (auch genannt Anwendungs- oder Anwendungsfallszenarien) beschreiben, wie das System zur Laufzeit auf einen bestimmten Auslöser reagieren soll. Hierunter fallen auch Szenarien zur Beschreibung von Effizienz oder Performance. Beispiel: Das System beantwortet eine Benutzeranfrage innerhalb einer Sekunde.
* Änderungsszenarien beschreiben eine Modifikation des Systems oder seiner unmittelbarer Umgebung. Beispiel: Eine zusätzliche Funktionalität wird implementiert oder die Anforderung an ein Qualitätsmerkmal ändert sich.

**Motivation.**

Szenarien operationalisieren Qualitätsanforderungen und machen deren Erfüllung mess- oder entscheidbar.

Insbesondere wenn Sie die Qualität Ihrer Architektur mit Methoden wie ATAM überprüfen wollen, bedürfen die in Abschnitt 1.2 genannten Qualitätsziele einer weiteren Präzisierung bis auf die Ebene von diskutierbaren und nachprüfbaren Szenarien.

**Form.**

Entweder tabellarisch oder als Freitext.

# Risiken und technische Schulden

1. **Authentification Service** als Microservice implementieren
2. **Token Klasse weniger aufblähen** – Tokenklasse aufsplitten in handling und Token
3. **User steht derzeit in der Authorisation Struktur** – schwerer zu benutzen bei der Buch zu Besitzer Zuordnung.
4. **Bisher keine JUnits die über HTTP**, die die REST Schnittstelle testen -bisher nur händisch mithilfe Postman

# Glossar

|  |  |
| --- | --- |
| **Begriff** | **Definition** |
| Media Ressource | Haupteinstigspunkt mit Rest Schnittstelle |
|  |  |
|  |  |

**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 |
| *Media Ressource* | *Haupt Einstigspunkt mit Rest Schnittstelle* |
| *<Begriff-2* | *<Definition-2>* |