HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

INSTITUT
FÜR
SOFTWARE

QUALI-T – Webbasiertes Knowledge Management Tool für Architectural Analysis und Quality Attribute Elicitation

Bachelorarbeit

Technischer Bericht

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2015

Autoren:	Corina Honegger, Emre Avsar
Betreuer:	Prof. Dr. Olaf Zimmermann
Projektpartner:	Institut für Software, Rapperswil
Experte:	Dr. Daniel Lübke
Gegenleser:	Prof. Oliver Augenstein
Abgabedatum:	12. Juni 2015

Aufgabenstellung

Aufgabenstellung Bachelorarbeit
Emre Avsar, Corina Honegger

QUALI-T – Webbasiertes Knowledge Management Tool für Architectural Analysis und Quality Attribute Elicitation

1. Auftraggeber und Betreuer

Diese Bachelorarbeit wird in Zusammenarbeit mit dem HSR Institut für Software durchgeführt.

Betreuer HSR:

Prof. Dr. Olaf Zimmermann, HSR FHO, Partner Institut für Software, ozimmerm@hsr.ch

2. Ausgangslage

Die Aufgaben von Software-Architekten und -Architektinnen gliedern sich in Architectural Analysis (Requirements Engineering), Architectural Synthesis (Design-Arbeit) und Architectural Evaluation (Reviews). Während für Architectural Synthesis zahlreiche Notationen und Arbeitstechniken existieren, haben die anderen beiden Bereiche bisher weniger Aufmerksamkeit in Methodenforschung und Werkzeugentwicklung erhalten. Es gibt zwar zahlreiche allgemeine Requirements Engineering Tools; diese unterstützen die Architekturarbeit, z.B. die Erstellung und Analyse von SMART NFRs und Quality Attribute Scenarios, allerdings nur rudimentär. In agilen Praktiken wird zumeist auf Features in Form von User Stories fokussiert; eine Erweiterung auf Quality Stories wurde zwar bereits vorgeschlagen, eine systematische Ausarbeitung und eine Werkzeugunterstützung fehlen aber bisher.

Am IFS gibt es Vorarbeiten zu Architectural Synthesis (z.B. Management von Architekturentscheidungen und zugehörigen Planungstasks), die u.a. mit serverseitigem Java und Frameworks wie Play und MySQL, aber auch mit AngularJS und TypeScript arbeiten.

3. Ziele der Arbeit und Liefergegenstände

Ziel der Arbeit ist die Konzeption und prototypische Umsetzung a) einer Methode und Wissensbasis und b) eines webbasierten Werkzeugs zur systematischen Erhebung und Analyse von praxistauglichen Qualitätsattributen (NFRs) im Rahmen der Architectural Analysis (und, optional, Architectural Evaluation):

- Architektur und prototypische Implementierung des Werkzeugs; Anbindung verwandter, bereits existierender Systeme (z.B. IFS-Vorarbeiten und Taskmanagementsysteme wie JIRA, Rally, Redmine oder Trello); Plattformcharakter
- Strukturierungskonzepte und Qualitätsmetriken für Methode und Wissensbasis, z.B. Heuristiken für SMART Quality Attributes, beispielhafte Population für cloud-native Applications, Enterprise Resource Planning (ERP)-Systeme oder eine andere aktuelle technisch-fachliche Domäne (auf Basis von NFR-Katalogen, die vom Betreuer als Literatursammlung zur Verfügung werden, sowie eigener Recherchen)

sowie die „Anleitung: Dokumentation Studien- und Bachelorarbeite" inklusive Anhängen (jeweils auf dem HSR Intranet verfügbar) zu beachten. Zudem ist eine kurze Projektresultatdokumentation für das Wiki von Prof. Zimmermann erwünscht (ggfs. kurzes Video).

7. Termine (Quelle: HSR Intranet)

16.02.15	Beginn der Bachelorarbeit, Ausgabe der Aufgabenstellung durch die Betreuer
Mai 2015	Fotoshooting. Genauere Angaben erteilt die Kommunikationsstelle rechtzeitig.
05.06.15	Abgabe Kurzbeschreibung (Abstract für Diplomarbeitsbroschüre) an das Abteilungssekretariat. Die Studierenden senden per Email das A0-Poster zur Prüfung an ihren Examinator/Betreuer.
12.06.15	Abgabe des Berichtes an den Betreuer bis 12.00 Uhr. Fertigstellung des A0-Posters bis 12.00 Uhr. Abgabe der Posters im Studiengangsekretariat 6.113.
12.06.15	HSR-Forum, Vorträge und Präsentation der Bachelor- und Diplomarbeiten, 16 bis 18 Uhr
03.08. - 21.08.15	Mündliche BA-Prüfung

Allfällige weitere Termine sind am Sekretariat der Abteilung Informatik zu erfragen und sollten entsprechend in einem Sitzungsprotokoll dokumentiert werden.

8. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS-Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert. Siehe auch Modulbeschreibung der Bachelorarbeiten, http://studien.hsr.ch/allModules/24809_M_BAI14.html.

Für die Beurteilung ist der HSR-Betreuer verantwortlich.

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/6
2. Berichte (Abstract, Management Summary, technische u. persönliche Berichte) sowie Gliederung, Darstellung und Sprache der gesamten Dokumentation.	1/6
3. Inhalt*)	3/6
4. Mündliche Prüfung zur Bachelorarbeit	1/6

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit festgelegt.

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Bachelorarbeiten.

Rapperswil, den 18. 02. 2015

Prof. Dr. Olaf Zimmermann
Institut für Software
Hochschule für Technik Rapperswil

Die Wahl der Frameworks und Technologiekomponenten für Web Client, Java-Server und Persistenzschicht ist dabei Teil der Aufgabe; konzeptionelle Integrität mit den IFS-Vorarbeiten ist im Hinblick auf Wartbarkeit anzustreben, kann aber auch auf integrativem Weg erreicht werden (z.B. durch Nutzung und Definition von Web APIs und RESTful HTTP).

Wichtige Erfolgskriterien für die Arbeit sind Reife und Praxistauglichkeit der Konzepte in Werkzeugarchitektur und Methode, Benutzbarkeit und Erweiterbarkeit der Implementierung (Tool, Heuristiken) sowie Aussagekraft und einfache Verwendbarkeit der Wissensbasis und der Methode.

4. Unterstützung:

Die erwartete und effektiv erhaltene Unterstützung wird durch die Studenten in Sitzungsprotokollen definiert und im BA-Bericht dokumentiert.

5. Zur Durchführung

Mit dem HSR-Betreuer finden in der Regel wöchentlich Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf zu veranlassen.

Alle Besprechungen, bei denen eine Vorbereitung durch den Betreuer nötig ist, sind von den Studenten mit einer Traktandenliste vorzubereiten. Beschlüsse sind in einem Protokoll zu dokumentieren.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. Arbeitszeiten sind zu dokumentieren.

Die Spezifikation der Anforderungen geschieht durch die Studenten in Absprache mit dem Betreuer. Bei Disputen entscheidet der Betreuer in Rücksprache mit den Studenten über die definitiv für die Bachelorarbeit relevanten Anforderungen.

Vorstudie, Anforderungsdokumentation und Architekturdokumentation sollten im Laufe des Projektes mittels Milestone mit dem Auftraggebern und dem Betreuer in einem stabilen Zustand abgenommen werden. Zu den abgegebenen Arbeitsresultaten wird ein vorläufiges Feedback abgegeben. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

Die Rechte an den Ergebnissen der Bachelorarbeit werden in einer separaten Vereinbarung definiert (Standard-Regelung).

6. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen. Die zu erstellenden Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Die Dokumentation ist vollständig auf CD/DVD in zwei Exemplaren abzugeben. Bei der Projektdokumentation sind die „Allgemeine Informationen zu Studien- und Bachelorarbeiten"

Abstract

Im Rahmen des Requirements Engineering in Softwareentwicklungsprojekten werden zahlreiche nichtfunktionale Anforderungen als Quality Attributes (QA) erhoben. Diese Architekturarbeit gliedert sich in drei Phasen, in denen die QAs teilweise als Input, teilweise als Output fungieren. In Phase eins, der Architectural Analysis, betrachtet man den gesamten Systemkontext sowie die funktionalen Anforderungen und leitet daraus die QAs ab. Anschliessend verfeinert man in Phase zwei, der Architectural Synthesis, diese QAs und kann auf deren Basis Architekturentscheidungen treffen. In Phase drei, der Architectural Evaluation, verifiziert und prüft man die QAs schliesslich, vorzugsweise durch eine neutrale, externe Stelle.

Die drei Phasen Architectural Analysis, Architectural Synthesis und Architectural Evaluation sind ohne Hilfsmittel schwierig zu realisieren. Etablierte Hilfsmittel stellen die Utility Trees und Quality Attribute Scenarios des Software Engineering Institute und zahlreiche ähnliche statische Dokumentvorlagen dar. Jedoch gibt es auf dem Markt aktuell kein Werkzeug, welches den QA-Engineering-Prozess über alle drei Phasen unterstützt und die statischen QA-Dokumentvorlagen zum Leben erweckt. Das Ziel dieser Bachelorarbeit ist es zu untersuchen, welche Eigenschaften in einem derartigen Werkzeug sinnvoll unterstützt werden können, und einen Prototyp eines solchen QA-Management-Werkzeugs zu entwickeln.

Im vorliegenden Bericht erfassen wir zunächst über Personas die zentralen Aufgaben und Anforderungen in den drei Phasen der Architekturarbeit und leiten daraus Use Cases ab. Anschliessend fokussieren wir auf die QAs und leiten die wünschenswerten Werkzeugeigenschaften aus Literatur und eigenen Projektbeispielen ab. Der dritte Teil der Arbeit beinhaltet Design und Entwicklung einer passenden Software, um die Architectural Analysis, Synthesis und Evaluation zu unterstützen.

Das Ergebnis der Arbeit ist die Client/Server-Webapplikation QUALI-T, die serverseitig mit Play Framework und clientseitig mit AngularJS entwickelt wurde. In QUALI-T legt man QAs als Vorlagen an und gliedert diese in Kategorien. Um die Qualität dieser QAs sicherzustellen, beurteilt man sie mit Hilfe von Quality Properties. Diese bestehen in QUALI-T standardmässig aus den SMART-Kriterien, nach denen ein QA Specific, Measurable, Agreed Upon, Realistic und Time-bound sein sollte; projektspezifische Erweiterungen sind möglich. Es besteht zudem die Möglichkeit, über eine Programmierschnittstelle Detektoren einzubauen, welche den Text eines QAs überprüfen und Feedback dazu geben. Als Beispiel ist eine Blacklist für Wörter implementiert, welche auf eine qualitativ unzureichende (also unscharfe, mehrdeutige) Spezifikation des QAs hinweisen können.

Während der Entwicklung haben wir festgestellt, dass es viele weitere Features gibt, die den Prozess der Architectural Analysis erleichtern. Sobald man mit Vorlagen und Variablen als Platzhalter arbeitet, kann man auch eine detaillierte Statistik über die eingesetzten Werte erheben und somit die Erstellung von QAs verbessern. Wenn wie in unserem Tool mit QA-Vorlagen gearbeitet wird, kann beispielsweise auch eine detaillierte Statistik über

verwendete Werte erstellt werden. Man kann Mengen von QA-Vorlagen für bestimmte Softwareprojekttypen erstellen und somit die Qualität der QAs signifikant erhöhen. Zusätzlich könnte man über die APIs beispielsweise auch eine Rechtschreibprüfung anbinden. QUALI-T zeigt die Möglichkeiten derartiger QA-Textanalysen auf und bietet damit eine gute Ausgangslage für Weiterentwicklungen und Werkzeugerweiterungen.

Vorwort und Danksagung

Die vorliegende Bachelorarbeit ist im Rahmen des Bachelor-Studiengangs an der Hochschule für Technik in Rapperswil entstanden.

Unser erster Dank geht an Herrn **Prof. Dr. Olaf Zimmermann** für die Betreuung der Bachelorarbeit.

Ein besonderer Dank richtet sich an **Andrea Duttwiler** von der AdNovum Informatik AG für die Verifikation, grammatische Korrektur sowie die lyrisch konstruktiven Vorschläge, die wir unter anderem auch in die Arbeit einfliessen liessen.

Sowie auch an **Teddy Huber** von der AdNovum Informatik AG für den Input im Bereich User Experience.

Inhaltsverzeichnis

1 Management Summary

1.1 Ausgangslage

In den heutigen Softwareentwicklungsprojekten haben die Projektleiter längst erkannt, dass eine gute Analyse und Planung den Implementationsaufwand und potenzielle Komplikationen erheblich reduzieren kann. Die Analyse ist derweilen eine eigene Disziplin: Requirements Engineering. Ein wichtiger Aspekt des Requirements Engineering sind die funktionalen und die nichtfunktionalen Anforderungen. Die funktionalen Anforderungen beschreiben, was das Produkt tun soll, während die nichtfunktionalen Anforderungen das Wie (mit welchen Eigenschaften) definieren. Die nichtfunktionalen Anforderungen werden im Englischen als „Non Functional Requirements“ oder Quality Attributes (QAs) bezeichnet. Die funktionalen Anforderungen sind meistens klare Anforderungen des Auftraggebers und einfach zu erfassen. Die QAs hingegen stellen die Requirements Engineers vor eine grössere Herausforderung.

Um den Requirements-Engineering-Prozess zu unterstützen, gibt es das in der nachfolgenden Abbildung gezeigte Modell, das die Architekturdesign-Aktivitäten in drei Phasen gliedert [1]:

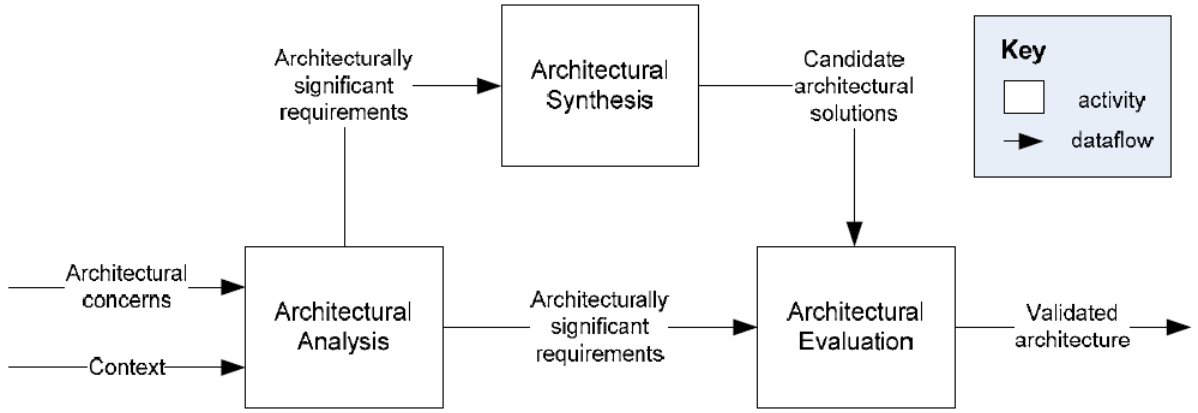


Abbildung 1 Architectural Design Activities [1]

In den einzelnen Phasen fungieren die QAs teilweise als Input, teilweise als Output: Zuerst betrachtet man den gesamten Systemkontext sowie die funktionalen Anforderungen und leitet in der Architectural Analysis daraus die QAs ab. Anschliessend verfeinert man diese QAs in der Architectural Synthesis und kann auf deren Basis bereits Architekturentscheidungen treffen. Während der Architectural Evaluation verifiziert man die QAs schliesslich, vorzugsweise durch eine neutrale, externe Stelle.

Aktuell gibt es auf dem Markt kein Tool, welches die Erstellung von QAs in allen drei Phasen aktiv unterstützt. Es existieren lediglich statische Dokumentvorlagen wie die Quality Attribute Scenarios des Software Engineering Institute oder sogenannte Utility Trees, die als Leitfaden dienen. Das Ziel dieser Bachelorarbeit ist es zu untersuchen, welche Eigenschaften in einem derartigen Werkzeug sinnvoll unterstützt werden können und einen Prototyp eines solchen QA-Management-Werkzeugs zu entwickeln.

1.2 Vorgehen und Technologien

Um die Anforderungen an solch ein Tool herauszufinden, haben wir uns zuerst überlegt, was für Jobprofile an einem Softwareentwicklungsprojekt beteiligt sind und welche Rollen diese in einem QA-Erstellungstool einnehmen können. Angelehnt an die drei Phasen der Architekturdesign-Aktivitäten haben wir uns für die Rollen Project Manager, Knowledge Catalog Curator, Architectural Analyst, Architectural Synthesizer und Architectural Evaluator entschieden und für diese Personas verfasst. Basierend auf diesen Personas haben wir in Zusammenarbeit mit unserem Betreuer und gestützt auf eigene Erfahrungen die funktionalen und nichtfunktionalen Anforderungen abgeleitet.

Bei der technischen Evaluation haben wir uns an Tools und Frameworks orientiert, die bereits im Institut für Softwareentwicklung der HSR im Einsatz sind. Das Resultat ist eine Client/Server-Webapplikation mit einer eigenständigen RESTful HTTP API. Das Frontend ist mit AngularJS und Bootstrap erstellt, während im Backend mit dem Play Framework in Java gearbeitet wird. Für den Persistence Level wird eine PostgreSQL-Datenbank verwendet.

1.3 Ergebnisse

Das Ergebnis ist die Webapplikation QUALI-T: **Qu**ality **A**ttribute E**li**citation **T**ool. Sie ermöglicht dem Benutzer, Quality Attribute Templates anzulegen. Diese können Variablen beinhalten, die während der späteren Benutzung in einem konkreten Softwareentwicklungsprojekt mit den passenden Werten befüllt werden. Es können auch Vorgabewerte für Variablen hinterlegt werden. Die Quality Attributes können ausserdem kategorisiert werden. Standardmässig passiert dies über die offiziellen Kategorien aus dem ISO 25010 Standard, alternativ kann der Benutzer auch eigene erstellen. Mehrere QA Templates können in einem Catalog zusammengefasst werden. Alle diese Schritte werden vom Knowledge Catalog Curator ausgeführt.

Anschliessend kommt der Architectural Analyst zum Einsatz. Mit Hilfe der Catalogs und den enthaltenen Templates erstellt er im Rahmen eines Projekts einen Satz von QAs. In diesem Schritt werden die Variablen durch passende Werte ersetzt, der Text überarbeitet und auf das Projekt angepasst. Durch das Verwenden von Vorlagen kann die Qualität der QAs bereits erhöht werden. Eine weitere Massnahme sind die Quality Properties: Eigenschaften, welche ein QA besitzen muss, damit es als „qualitativ hochwertig“ – spezifisch, messbar und realistisch – deklariert werden kann. QUALI-T bietet hierfür die SMART Quality Properties. Die fünf Buchstaben SMART stehen für die QA-Eigenschaften Specific, Measurable, Agreed Upon, Realistic und Time-bound. Auch dieses können nach Belieben durch eigene Properties ergänz werden.

Die Aufgabe des Architectural Synthesizer und des Evaluator ist zu überprüfen, ob die definierten QAs die Quality Properties erfüllen.

Die drei Schritte Quality Attribute Template, Catalog und Project erstellen sind in der folgenden Abbildung mit Bildern aus QUALT-T dargestellt:

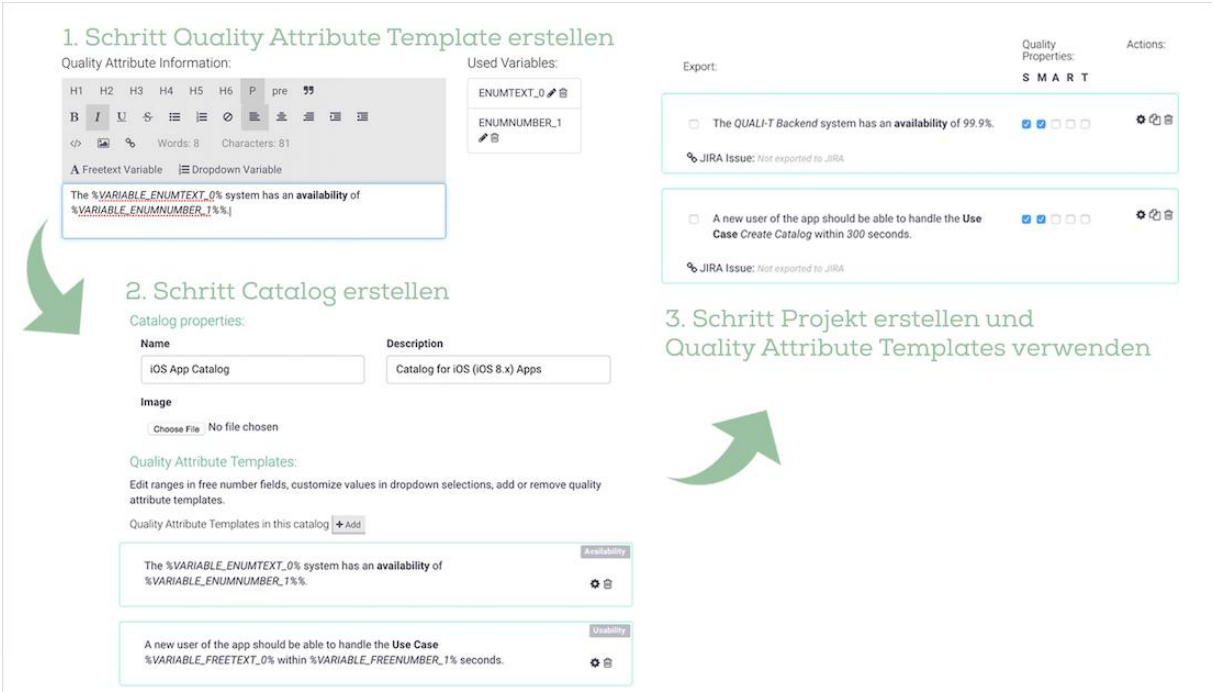


Abbildung 2 Workflow in QUALI-T zur Erstellung von QAs

Um die Qualität der QAs weiter zu erhöhen, gibt es eine Fuzziness Detector API. Über diese können sogenannte Detectors angeschlossen werden. Ein Beispiel hierfür ist der implementierte Blacklist Detector. Er überprüft den Text auf Schlüsselwörter, welche auf eine unzureichende Qualität hinweisen können. QUALI-T informiert den Benutzer über die gefundene Schwachstelle und bietet gleichzeitig einen Lösungsvorschlag an. Weitere solcher Detektoren können bei Bedarf angeschlossen werden. Zusätzlich zur Fuzziness Detector API werden über die eingefügten Variablenwerte Statistiken geführt. Im Projekt kann der Benutzer somit den tatsächlichen Wert mit der Statistik vergleichen.

Sind die QAs einmal erfasst, können sie via Knopfdruck in ein JIRA-Projekt exportiert oder als Report in Form einer PDF- oder XML-Datei heruntergeladen werden.

1.4 Ausblick

QUALI-T ist ein Prototyp, welcher aufzeigt, wie man QAs systematisch qualitativer formulieren kann. Um den QA-Erstellungsprozess zu unterstützen, kann man weitere Detectors entwickeln und über die Fuzziness Detector API anschliessen, beispielsweise eine Rechtschreibeprüfung. Eine andere Erweiterungsoption besteht darin, das Tasks-System auszubauen, sodass User sich gegenseitig Tasks zuweisen können, welche auf die Rolle der User in einem Projekt Bezug nehmen. Durch die Funktionalität des Catalog-Imports/-Exports besteht auch die Möglichkeit, themenbezogene Catalogs zu erstellen und diese kommerziell als Erweiterung anzubieten. Auch die Statistik eignet sich gut für weitere Funktionalitäten. Diese Funktionen können gemäss den Kundenwünschen ausgebaut und angepasst werden. Da das Front- und das Backend eigenständig sind, könnte wahlweise auch die eine oder andere Komponente ausgetauscht werden.

2 Problemanalyse

In diesem Kapitel wird das Ziel unserer Bachelorarbeit (BA) genauer vorgestellt und die originale Aufgabenstellung ergänzt und abgerundet.

2.1 Definitionen

Im Folgenden sind einige Definitionen beschrieben, welche in diesem Dokument vorkommen und deren Bedeutung der Leser verstehen sollte.

2.1.1 Funktionale Anforderungen vs. nichtfunktionale Anforderungen

Unter Anforderungen versteht man grundsätzlich die zu erfüllende Eigenschaft oder Leistung eines Projektergebnisses (Software, System). Sie können in funktionale Anforderungen (FA) und nichtfunktionale Anforderungen (NFA) unterteilt werden. Während die FAs beschreiben, *was* das Produkt tun soll, befassen sich die NFRs mit dessen Eigenschaften, das heisst, *wie* die FAs erfüllt werden sollen. NFRs umschreiben viel mehr die Umstände (engl. Constraints) und somit die Qualität, unter der die FAs erfüllt werden sollen. Oftmals werden auch die englischen Varianten „Functional Requirement“ (FR) und „Non Functional Requirements“ (NFR) verwendet.

2.1.2 Quality Attribute (QA)

In unserer Dokumentation sowie im Programm QUALI-T wird der Begriff „Quality Attribute“ gleichbedeutend mit „nichtfunktionale Anforderung“ verwendet. Die genaue Definition ist daher bereits im Kapitel 2.1.1 enthalten.

2.1.3 Quality Attribute Trees

Um herauszufinden, welche QAs in einem Projekt wichtig sind, gibt es verschiedene Ansätze und Hilfsmittel. Im SEI-ATAM-Prozess erstellt man dazu beispielsweise sogenannte Quality Attribute Utility Trees. Diese helfen die richtigen QAs zu erstellen, indem man über Szenarien geht und diese gewichtet. [2][1] SEI stellt im Zusammenhang mit Quality Attributes auch selber Trees zur Verfügung, welche eine Kategorisierung der QAs erlauben. [3][2]

2.1.4 Create, Read, Update, Delete, Search (CRUDS)

CRUDS ist in der Informatikfachsprache eine gängige Abkürzung, um diverse Aktivitäten zusammenzufassen: Create, Read, Update, Delete und Search. Dies sind die Standardaktionen, die man auf ein Objekt anwenden kann. Der Begriff ist vor allem auch im Zusammenhang mit Datenbankprozeduren geläufig. Wir verwenden bei den Use Cases in unserer BA diese Kurzbeschreibung von Benutzeraktionen als geläufigen Ausdruck.

2.1.5 Specific, Measurable, Agreed Upon, Realistic, Time-bound (SMART)

SMART ist ein in zahlreichen Fachgebieten und Berufsgruppen verbreitetes Akronym, welches 1981 zum ersten Mal von George T. Doran im Kontext von Personalführung dokumentarisch belegbar eingeführt wurde [4]. Es beschreibt, wie Aufgaben und Ziele (Objectives and Goals) formuliert sein sollten. Mündlich wurde der Begriff allerdings schon früher von einem Paul J. Meyer verwendet. In

[1] Quelle: http://www.sei.cmu.edu/reports/00tr004.pdf
[2] Quelle: http://www.sei.cmu.edu/reports/95tr021.pdf

dieser Version stehen die Buchstaben für folgende Eigenschaften: **S**pecific, **M**easurable, **A**ttainable, **R**ealistic und **T**angible [5].
Über die Bedeutung der Buchstaben S und M ist man sich einig. Bei A, R und T gibt es verschiedene Begriffe, die Kernaussage ist jedoch bei allen die Gleiche. Das Ziel unserer BA ist ein Tool, welches dabei hilft, die QAs in einem Projekt SMART zu machen. Ein QA ist nur ein gutes QA, wenn es auch alle fünf SMART-Kriterien erfüllt[3]. In unserem Kontext haben wir uns für die folgende Bedeutung des Akronyms entschieden:

Specific	Spezifisch, genau
Measurable	Mess- und überprüfbar
Agreed Upon	Das Team kann mit verfügbarem Know-how die Aufgabe erfüllen oder das Ziel erreichen
Realistic	Nach heutigem Stand der Technik realistisch
Time-bound	Terminiert, mit den verfügbaren Ressourcen in definiertem Zeitraum möglich

2.2 Anforderungsanalyse

In diesem Kapitel fassen wir unsere detaillierte Analyse der Aufgabenstellung zusammen. Um sicherzustellen, dass wir ein gemeinsames Verständnis der konkreten Aufgabe haben, hielten wir mehrfach Rücksprache mit unserem Betreuer, der zugleich Vertreter unserer Zielgruppe ist. Nebst den nachfolgenden Abschnitten erstellten wir aus der Anforderungsanalyse auch die Projektplanung gemäss Kapitel **Error! Reference source not found.**.

2.2.1 Allgemeine Beschreibung

2.2.1.1 Produktperspektive

Auf dem Markt existieren diverse Tools, welche die Software Engineers und die Analysten während der Anforderungsanalyse bei der Erstellung von FRs zur Festlegung des Funktionsumfangs unterstützen. Es ist in der Softwareentwicklung allgemein bekannt, wie wichtig FRs als Planungs- und Designgrundlagen sind. Jedoch trifft man bei Projekten immer wieder auf unzureichend spezifizierte NFRs. Dies liegt zum einen daran, dass viele gar nicht wissen, was gute NFRs sind oder wie man diese systematisch erstellt. Zum andern sind die QAs aber auch regelmässig schlecht formuliert, sprich nicht **SMART.** Damit sind Kriterien gemeint, die nicht spezifisch genug (**S**pecific) und vor allem nicht messbar (**M**easurable) sind. Zum Beispiel die Anforderung „Das System muss hochverfügbar sein.“.

Welches System soll hochverfügbar sein? Und was heisst hochverfügbar? Dieses Wort ist nicht messbar ohne eine konkrete Zahl. Hier kommt QUALI-T zum Einsatz. Die QUALI-T Software soll den ganzen QA-Erstellungsprozess begleiten und den Anwender bei der Erstellung von *SMART*en NFRs unterstützen.

[3] In der Softwarearchitekturausbildung werden beispielsweise die SMART-Kriterien verwendet, um die NFRs zu präzisieren und quantifizieren. [18]

2.2.1.2 Produktbeschreibung

Anschliessend ist ein Überblick der Domäne als Domainmodel zu sehen. Detailliertere Beschreibungen zu den Objekten in der Domäne sind unter der Abbildung zu finden.

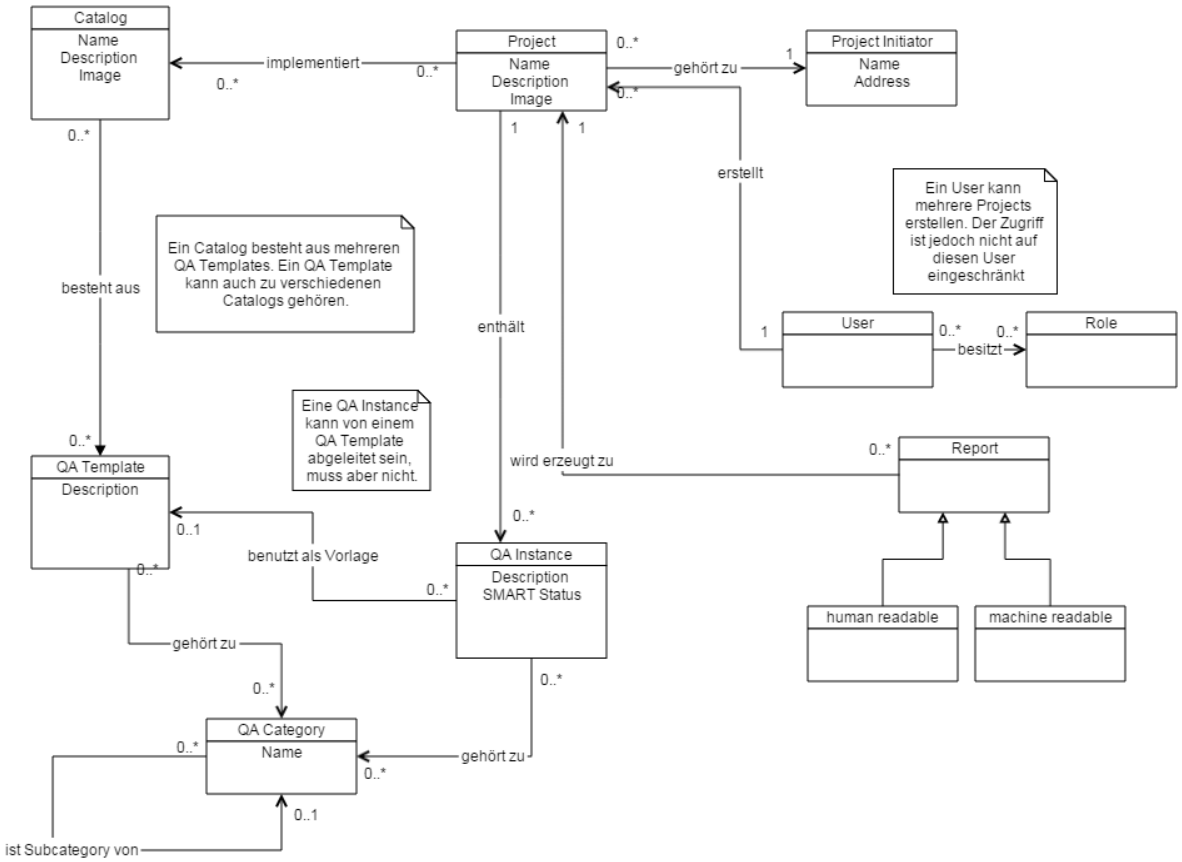


Abbildung 3 QUALI-T-Domainmodel

In QUALI-T können *Catalogs* erstellt werden. Diese bestehen aus mehreren konkreten *QA Templates*. Wird ein neues *Project* angelegt, wählt der Anwender zuerst einen beliebigen *Catalog* aus. Anschliessend sieht er alle *QA Templates* von diesem *Catalogs* und kann jene auswählen, welche für das Projekt relevant sind. Er kann jedes *QA Template* beliebig anpassen, womit es zu einer *Instance* wird. Alternativ kann er auch manuell *Instances* ohne Vorlage anlegen. Es ist möglich, einem *Project* einen *Project Initiator* zuzuordnen, was die Verwaltung der *Projects* erleichtert. Im Folgenden werden die Begriffe anhand eines Beispiels genauer erklärt:

Catalog
Name: Cloud Solutions QAs

QA Template
Description: Die Verfügbarkeit der Softwarekomponente „_____“ muss zu __% gewährleistet sein.

Project
Title*:* Neuer Webshop-Musterkunde

Project Initiator
Name: Musterkunde
Location: Musterhausen

Instance
Description: Die Verfügbarkeit der Softwarekomponente „Web Frontend“ muss zu 99% gewährleistet sein.

2.2.1.3 Einschränkungen

Da wir diese Bachelorarbeit für das Institut für Softwareentwicklung (IFS) realisieren, gibt es nicht viele Einschränkungen. Einzig bei der Programmiersprache haben wir die Weisung, mindestens backend-seitig in Java zu setzten. Das fertige Softwareprodukt sollte möglichst mit den anderen Tools des IFS kompatibel sein (Beispiel EPPII oder AD-REPO). Diese Kompatibilität könnte mit einer RESTful HTTP API abgedeckt werden. Denn eine API erlaubt denn Zugriff auf die Backend-Logik aus jedem beliebigen Tool oder Web-GUI heraus.

2.2.1.4 Schnittstellen und Abhängigkeiten

Während der Analyse der Umsysteme und der Schnittstellen haben wir das folgende Systemkontextdiagramm erarbeitet. Es zeigt auf, welche Verbindungen in Frage kommen können. Die Wahl, welche Systeme tatsächlich angebunden werden, wird zu einem späteren Zeitpunkt durchgeführt und im Kapitel 3.1.12.2 beschrieben.

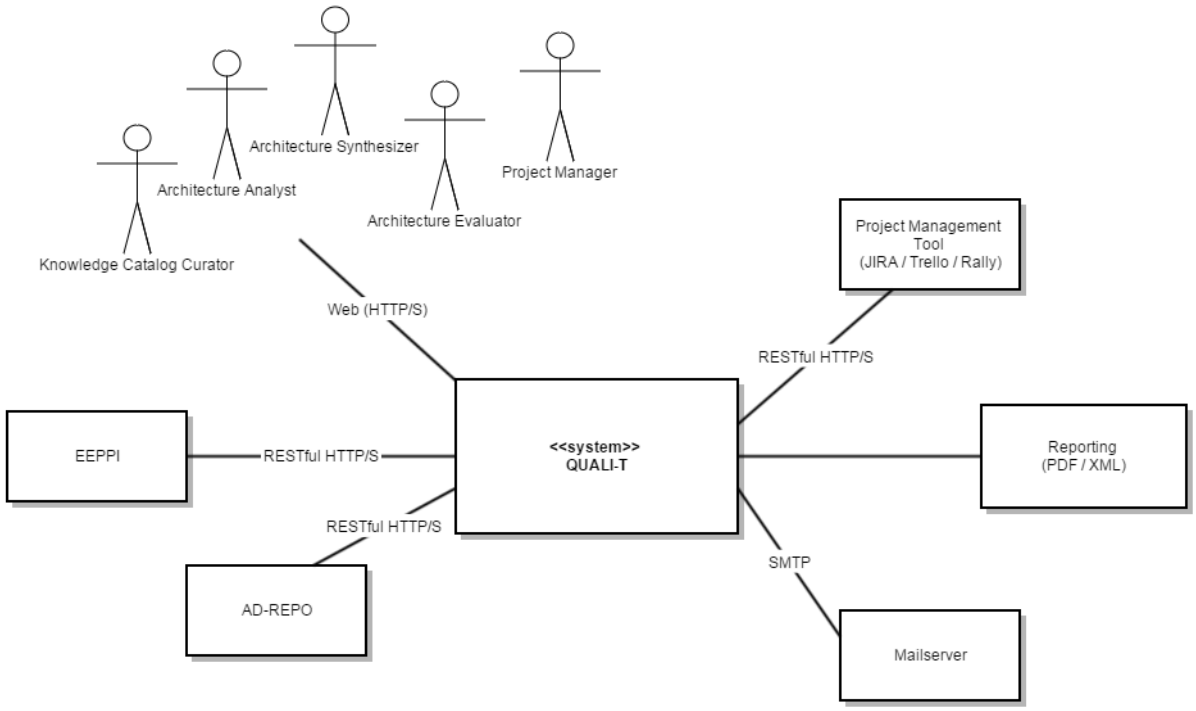


Abbildung 4 Systemkontextdiagramm mit möglichen Verbindungen zu verschiedenen Services und Hosts

2.2.2 Funktionale Anforderungen

In den nachfolgenden Kapiteln werden an Hand von Personas und Use Cases die funktionalen Anforderungen beschrieben.

2.2.2.1 Aktoren und Stakeholder

Um die potenziellen Aktoren von QUALI-T besser zu analysieren, haben wir diese in Rollen aufgeteilt. Für jede Rolle ist nachfolgend eine Personenbeschreibung aufgeführt, welche in der Fachsprache als Persona bezeichnet wird.

Persona/Rolle	**Project Manager**
Name	Susanne Grimm

Alter	46 Jahre
Beruf, Ausbildung	Projektleiterin (PL), langjährige Erfahrung als PL in Industrie, Marketing und IT
Tägliche Aufgaben	Übernimmt in grossen Projekten die Gesamtprojektleitung und unterstützt die Koordination und die Kommunikation mit den Kunden aktiv, akquiriert neue Kunden und Projekte
Likes	Koordinieren und Kommunizieren sind ihre Lieblingsbeschäftigungen.
Pain Points	Das dauernde Nachfragen bei den Softwareentwicklern und -architekten über den Status der NFRs und FRs nervt sie. Dies erschwert ihr das Controlling und sie hinterlässt beim Kunden ein schlechtes Image.
Ziele	Sie will den Status der NFR-Verifikation und -Implementation prüfen und die finalen Versionen für ihre Projektplanung und für das Projektmanagement-Tool exportieren können.

Persona/Rolle	**Knowledge Catalog Curator**
Name	Heinz Fisherman
Alter	29 Jahre
Beruf, Ausbildung	IT-Projektleiter, Grundlagenwissen über Software Engineering durch Informatiklehre und anschliessende Weiterbildung zum PL
Tägliche Aufgaben	Fungiert als Schnittstelle zwischen Kunde und Softwareabteilung, plant und koordiniert die Softwareprojekte
Likes	Gespräche mit Kunden und die Durchplanung von Projekten „im Kopf“ sind seine absoluten Lieblingsaufgaben.
Pain Points	Dokumentieren der Planungsaktivitäten hingegen ist ein notwendiges Übel.
Ziele	Er will NFRs, die er aus seiner Erfahrung mit dem Kunden oder bestimmten Projekttypen (z.B. Cloud-Projekte, Webapplikationen) kennt, als Template für seine Teamkollegen zur Verfügung stellen. Dies soll Zeit sparen und dabei helfen, dass nichts vergessen geht, was er während der späteren Projektlaufzeit durch Planänderungen korrigieren muss. Ausserdem will er sicherstellen, dass die NFRs möglichst vollständig und qualitativ formuliert werden.

Persona/Rolle	**Architecture Analyst**
Name	Tom Eisenhauer
Alter	36 Jahre
Beruf, Ausbildung	Senior .NET-Entwickler und Teamleiter, arbeitet seit vielen Jahren in der Softwareentwicklung und übernimmt dank seiner Erfahrung öfters auch die Funktion als Requirements Engineer in grösseren Projekten.
Tägliche Aufgaben	Einen grossen Teil seiner Arbeitszeit verbringt er mit Programmieren und der technischen Unterstützung seiner Teammitglieder. Bei grösseren Projekten erstellt er auf Grundlage der Informationen des Projektleiters und der Use Cases auch den ersten Entwurf der FR und NFRs.
Likes	Tom ist ein Programmierer mit Herzblut, Programmieren ist seine wahre Passion.
Pain Points	Er unterstützt seine Teammitglieder zwar gerne, aber den Administrativaufwand seiner Teamleiterfunktion würde er gerne abgeben.

Ziele	Für die Erstellung der FRs hat er bereits ein ausgefeiltes template- und werkzeuggestütztes Vorgehensmodell, aber ihm fehlt eines für die NFRs. Diese muss er mühsam anhand verschiedener Vorlagedokumente jedes Mal neu zusammenstellen. Und sind sie einmal erstellt, werden sie in der Regel auch schon von allen Projektbeteiligten vergessen. Er wünscht sich die Möglichkeit, die Vorlagedokumente an einem zentralen Ort zu verwalten und zu verfolgen.

Persona/Rolle	**Architecture Synthesizer**
Name	Priska Hofmann
Alter	27 Jahre
Beruf, Ausbildung	Java-Entwicklerin, Fachhochschulstudium Informatik
Tägliche Aufgaben	Sie ist im Scrum Team tätig und ihre Hauptaufgabe ist das Programmieren. Sie sitzt auch mit dem Projektleiter und/oder mit den Kunden zusammen und versucht mit Hilfe der Anforderungen die bestmögliche Softwarearchitektur zu planen.
Likes	Ihr gefällt vor allem die Abwechslung der zu entwickelnden Funktionen, die sie durch die Arbeit im Scrum Team hat.
Pain Points	Unklar definierte Anforderungen sind leider eine alltägliche Tatsache. Die ständig notwendigen Rückfragen beim Projektleiter und beim Kunden während der Entwicklung findet sie nervig und anstrengend. Stattdessen wünscht sie sich, die von den Architecture-Analysten oftmals unter- oder überspezifizierten NFRs schon vor der Programmierarbeit optimieren zu können.
Ziele	Sie wünscht sich klar definierte Anforderungen bei den Projekten, die sie bei Bedarf auch selber ergänzen oder spezifizieren kann. Diese Anforderungen möchte sie so ausarbeiten, dass sie anschliessend sinnvolle Architecture Decisions (AD) treffen und durch die NFRs ausreichend begründen kann.

Persona/Rolle	**Architecture Evaluator**
Name	Simon Huber
Alter	31 Jahre
Beruf, Ausbildung	Software Architect, ETH-Masterstudium in Softwarearchitektur
Tägliche Aufgaben	Neben seinen Programmieraufgaben übernimmt er regelmässig das Verifizieren von Architecture Designs und Architecture Requirements fremder Projekte.
Likes	Da er regelmässig den Auftrag erhält, NFRs und FRs zu verifizieren, hat er ein sehr ausgewogenes Tätigkeitsfeld zwischen Analyse und praktischer Entwicklung. Dies motiviert ihn enorm.
Pain Points	Ihm fehlt bei schlecht formulierten NFRs die Messlatte zum Überprüfen des Architecture Designs. Er muss Annahmen treffen und ständig nachfragen, wodurch oftmals ein E-Mail-Ping-Pong entsteht. Dadurch verliert er die Übersicht und ihm entgehen manchmal wichtige Anmerkungen.
Ziele	Eine Möglichkeit zur Absicherung und Nachvollziehung seiner Review-Tätigkeiten würde ihm ein viel sichereres Gefühl geben und Struktur in diese Tätigkeit bringen.

Zwischen den verschiedenen Aktoren gibt es Interaktionspunkte. Der nachfolgende Flowchart zeigt eine Möglichkeit auf, wie ein Zusammenspiel der verschiedenen Rollen bei einem NFR-Projekt aussehen könnte, und wie QUALI-T die Schritte unterstützt. Die Use Cases sind bewusst knapp zusammengefasst oder weggelassen. Der Flowchart soll lediglich verdeutlichen, welche Schnittstellen zwischen den Aktoren vorhanden sind.

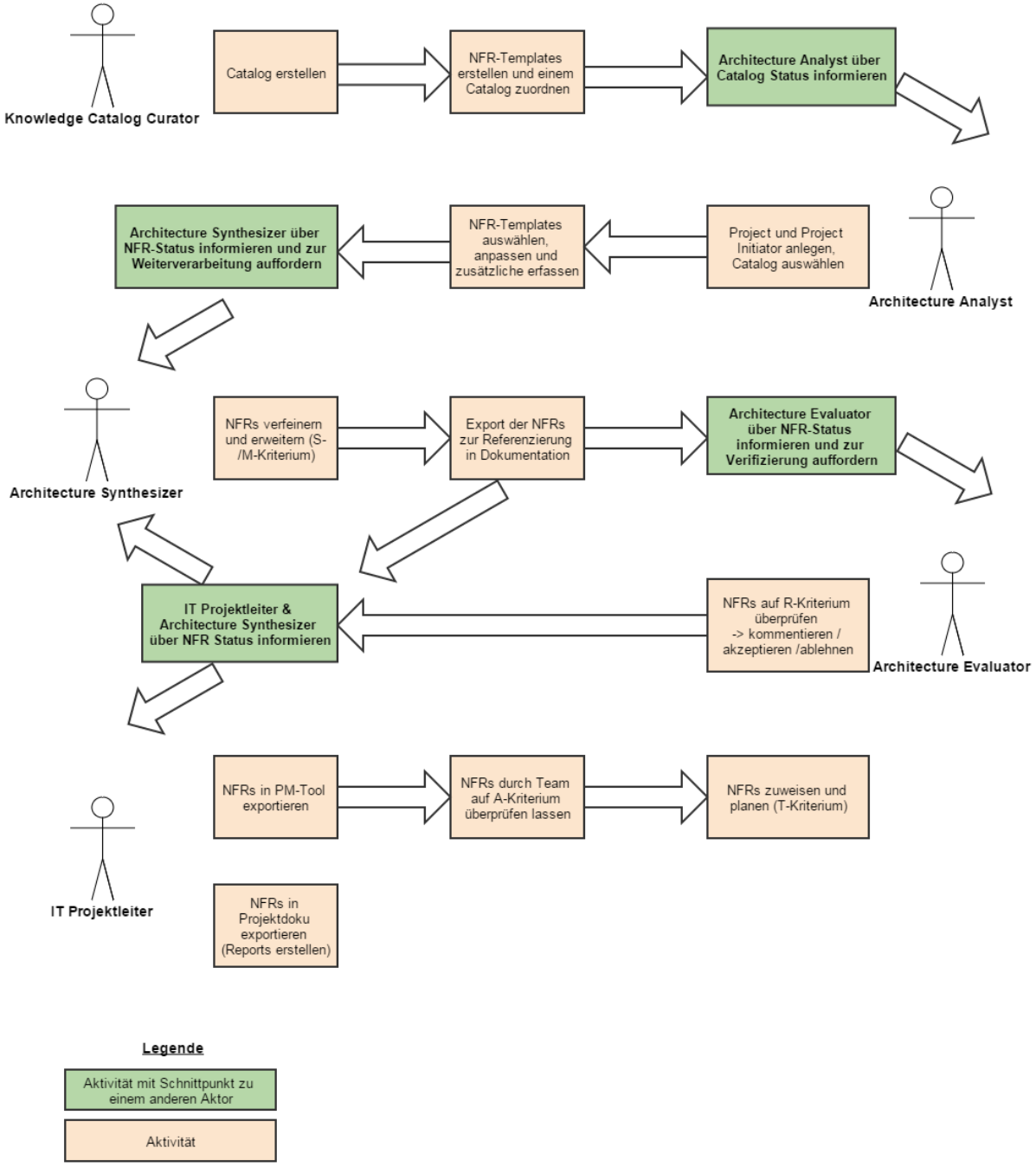


Abbildung 5 Aktoren-Interkations-Flowchart

2.2.2.2 Use Cases

Es gibt 19 Use Cases, welche die aufgeführten Aktoren und Stakeholder ausführen möchten. Die wichtigsten und komplexeren sind nachfolgend detailliert aufgeführt, zu allen anderen ist eine kompakte Beschreibung im „Brief Case“-Format vorhanden.

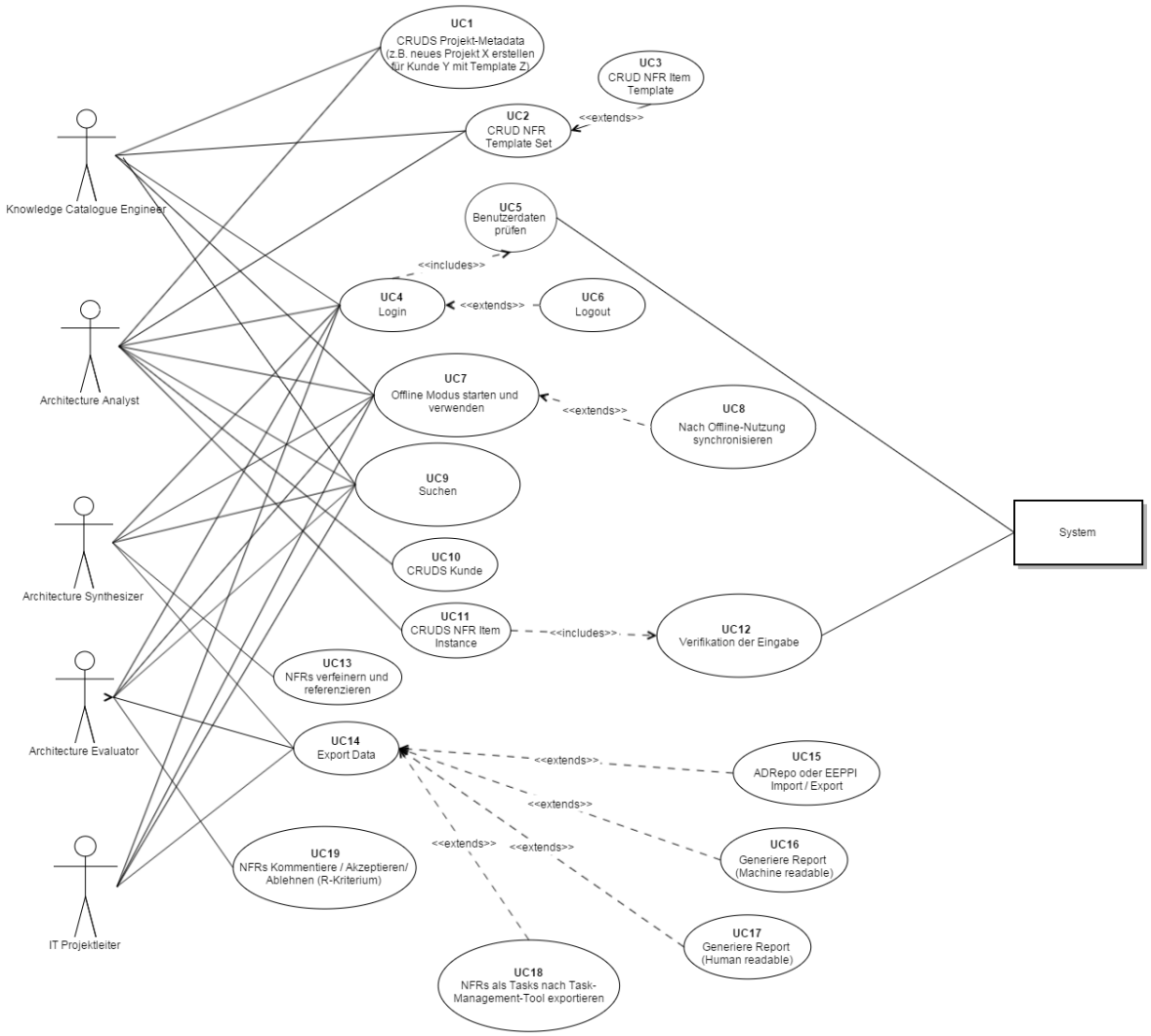


Abbildung 6 Use-Case-Diagramm für QUALI-T

Die Tabelle unten zeigt alle Use Cases und die dazugehörige Priorisierung, welche für die Construction Phase dieser BA von grosser Bedeutung ist. Der Fokus wird daraufgelegt, alle Use Cases mit der Priorität 1 (zwingende Grundfunktionen) und möglichst viele mit Priorität 2 umzusetzen. Falls anschliessend noch Zeit bleibt, werden die Use Cases mit der Priorität 3 in Angriff genommen.

Nr.	Priorität **[1 = required]** **[2 = nice to have]** **[3 = optional]**	Titel
UC1	1	CRUDS Projekt-Metadaten
UC2	1	CRUDS NFR Template Set („Fully Dressed“)
UC3	1	CRUDS NFR Item Template
UC4	1	Login
UC5	1	Benutzerdaten prüfen
UC6	1	Logout
UC10	1	CRUDS Auftraggeber
UC11	1	CRUDS NFR Item Instance
UC12	2	Verifikation der Daten
UC13	2	NFRs verfeinern und referenzieren
UC14	2	Export Data
UC16	2	Generiere Report (human readable)
UC17	2	Generiere Report (machine readable)
UC19	2	NFRs kommentieren / akzeptieren / ablehnen (R-Kriterium)
UC7	3	Offline-Modus starten und verwenden
UC8	3	Nach Offline-Nutzung synchronisieren
UC9	3	Suchen
UC15	3	ADRepo- oder EEPPI-Import/-Export
UC18	3	NFRs als Task nach Taskmanagement-Tool exportieren

Tabelle 1 UC Priorisierung

2.2.2.2.1 UC1 – CRUDS Projekt-Metadaten

Der Aktor erfasst, bearbeitet oder löscht ein Projekt im Tool. Das Projekt wird mit einem Kunden und optional mit einem Template Set verknüpft.

2.2.2.2.2 UC2 – CRUDS NFR Template Set

Da der Create-Ablauf das Main Success Scenario dieses Use Cases darstellt, wird nur dieser "Fully Dressed" beschrieben.

ID	UC2
Title	Create NFR Template Catalog
Primary Actors	Knowledge Catalog Engineer
Preconditions	• Der Aktor ist angemeldet • Der Aktor besitzt die Berechtigungsrolle als Knowledge Catalog Engineer
Postconditions	• Das Template-Set ist im System vorhanden und abrufbar
Main Success Scenario	1. Der Aktor öffnet den Dialog zum Erstellen eines neuen Template-Sets. 2. Der Aktor wählt aus, in welcher *Template-Kategorie* er das Set erstellen will, setzt einen *Titel* und schreibt optional einen *Kommentar*. 3. Das System zeigt ihm alle vorhandenen NFR-Templates als Liste an. 4. LOOP Der Aktor fügt ein NFR-Template[4] zum Set hinzu. END LOOP 5. Der Aktor speichert das Template-Set ab. 6. Das System prüft, ob alle zwingenden Eingaben gemacht sind (*Titel* und *Template-Kategorie).* 7. Das System speichert das Template und zeigt die Template-Set-Übersicht-Liste an.
Extensions (E)	E4 1. Der Aktor erstellt ein neues NFR-Template. *(Tabelle 2 UC2 - Create NFR Template Catalog im Format "Fully Dressed"* UC3 – CRUDS NFR Item Template) 2. Das System fügt das NFR-Template zur Liste aus Punkt 3 hinzu. E5 1. Fehlen notwendige Angaben, erhält der Aktor eine Fehlermeldung, kann die Angaben ergänzen und das Speichern wiederholen.
Alternate Flow (A)	-
Exceptions (Ex)	Ex(5) 1. Das System erkennt, dass nicht alle zwingenden Eingaben gemacht wurden, und zeigt an, wo was fehlt. 2. Das System fordert den User mittels Fehlermeldung auf, die fehlenden Eingaben zu ergänzen.
Frequency of Use	Regelmässig
Priority	Hoch

Tabelle 2 UC2 - Create NFR Template Catalog im Format "Fully Dressed"

2.2.2.2.3 UC3 – CRUDS NFR Item Template

Da der Create-Ablauf das Main Success Scenario dieses Use Cases darstellt, wird nur dieser "Fully Dressed" beschrieben.

[4] Ein Template-Set besteht aus mehreren NFR Templates. Ein NFR Template ist eine konkrete NFR, die als Vorlage erfasst wurde.

ID	UC3
Title	CRUDS NFR Item Template
Primary Actors	Knowledge Catalog Curator
Preconditions	• Der User ist angemeldet • Der User besitzt die Berechtigungsrolle als Knowledge Catalog Curator
Postconditions	• Das NFR Item Template ist persistiert und dem Standard-Catalog zugeordnet
Main Success Scenario	1. Der Aktor öffnet den Dialog zum Erstellen eines neuen NFR Item Template. 2. Der Aktor wählt aus, zu welchen Kategorien das NFR Item Template gehört. 3. Der Aktor schreibt die *Description* (NFR Text). 4. LOOP Der Aktor fügt eine Variable hinzu. ENDLOOP 5. Der Aktor speichert das neue NFR Item Template ab. 6. Das System prüft, ob alle zwingenden Eingaben gemacht sind (*Description).* 7. Das System speichert das NFR Item Template und zeigt die Übersichtsliste mit allen NFR Item Templates an.
Extensions (E)	E(4) 1. Das System öffnet den Dialog zum Hinzufügen einer Variable 2. Das Aktor wählt den Variabel-Typ aus. 3. Der Aktor gibt alle für den gewählten Variabel-Typ relevanten Daten ein. 4. Der Aktor klickt auf die Schaltfläche „Variabel hinzufügen“ 5. Das System fügt die Variabel zur NFR Item Template *Description* hinzu und kehrt zum „NFR Item Template Erstellen“-Dialog zurück.
Alternate Flow (A)	-
Exceptions (Ex)	Ex(6) 1. Das System erkennt, dass nicht alle zwingenden Eingaben gemacht wurden, und zeigt an, wo was fehlt. 2. Das System fordert den User mittels Fehlermeldung auf, die fehlenden Eingaben zu ergänzen.
Frequency of Use	Wöchentlich
Priority	Hoch

Tabelle 3 UC3 - CRUDS NFR Item Template im Format "Fully Dressed"

2.2.2.2.4 UC4 – Login

Ein Aktor öffnet die Weboberfläche und gibt seinen Benutzernamen und sein Passwort ein. Entsprechend seiner Berechtigungsrollen wird die View geladen, welche ihm Zugriff auf alle dazugehörigen Funktionen erlaubt.

2.2.2.2.4.1 <<includes>> UC5 – Benutzerdaten prüfen

Das System überprüft die eingegebenen Benutzerdaten und prozessiert bei erfolgreicher Authentifizierung die Anmeldung. Sind die Daten falsch, wird eine Fehlermeldung übermittelt.

2.2.2.2.4.2 <<extends>> UC6 – Logout

Ein Aktor klickt auf den „Logout“-Button. Seine Session wird ungültig gemacht. Der Aktor muss sich beim nächsten Aufruf der Webapplikation neu anmelden.

2.2.2.2.5 UC7 – Offline-Modus starten und verwenden

Ein Aktor startet das Tool ohne Verbindung zum Server. Es wird eine Offline-Version gestartet, die die zuletzt synchronisierten Daten anzeigt. Der Aktor kann beliebige Änderungen vornehmen, die temporär lokal gespeichert werden.

2.2.2.2.5.1 UC8 – Nach Offline-Nutzung synchronisieren

Ein Aktor ist wieder im gleichen Netz wie der Server und synchronisiert seine lokale Version mit der Server-Version. Treten Synchronisierungskonflikte auf, wird der User dazu aufgefordert, diese manuell zu beheben, um keine Daten zu verlieren.

2.2.2.2.6 UC9 – Suchen

Der Aktor gibt einen beliebigen Ausdruck ins Suchfeld ein. Das System liefert ihm alle mittels Volltextsuche gefundenen Ergebnisse. Durchsucht wird z.B. nicht nur der Titel der NFRs, sondern auch die komplette Beschreibung.

2.2.2.2.7 UC10 – CRUDS Auftraggeber

Der Aktor startet die Kundenverwaltung des Tools und führt dort Verwaltungsaktionen auf die Kundendaten aus.

2.2.2.2.8 UC11 – CRUDS NFR Item Instance

Da der Create-Ablauf das Main Success Scenario dieses Use Cases darstellt, wird nur dieser "Fully Dressed" beschrieben.

ID	**UC11**
Title	CRUDS NFR Item
Primary Actors	Architecture Analyst
Preconditions	• Der User ist angemeldet • Der User besitzt die Berechtigungsrolle als Architecture Analyst • Ein Projekt wurde gemäss UC Projekt-Metadata angelegt
Postconditions	• Das NFR Item ist persistiert und einem Projekt zugeordnet
Main Success Scenario	8. Der Aktor wählt ein Projekt aus. 9. Das System zeigt eine Liste mit allen dem Projekt zugeordneten NFR Items. 10. Der Aktor klickt auf eine Schaltfläche „Neues NFR Item von Template“. 11. Das System zeigt alle dem Projekt zugeordneten NFR Templates. 12. Der Aktor wählt ein NFR Template. 13. Das System öffnet einen Editor mit dem NFR Template als Vorlage. 14. Der Aktor passt die NFR-Definition gemäss seiner Vorstellungen an. 15. Der Aktor klickt auf die Schaltfläche „Speichern“. 16. Das System speichert das NFR Item und kehrt zur Liste von Punkt 2 zurück.
Extensions (E)	E(8)/E(A3.4) 1. Das System verifiziert das NFR Item gemäss UC *Verifikation der Daten*.
Alternate Flow (A)	A(3) 1. Der Aktor klickt auf die Schaltfläche „Neues NFR Item“ 2. Das System öffnet einen Dialog mit einem Textfeld *NFR Description.* 3. Der Aktor schreibt das NFR in das NFR *Description* Feld. 4. Der Aktor klickt auf die Schaltfläche „Speichern“. 5. Das System speichert das NFR Item und kehrt zur Übersichtsliste aller Projekt-NFR-Items zurück.
Exceptions (Ex)	-
Frequency of Use	Mehrmals täglich
Priority	Hoch

Tabelle 4 UC11 - CRUDS NFR Item im Format "Fully Dressed"

2.2.2.2.8.1 <<includes>> UC12 – Verifikation der Daten

Das System überprüft, ob die in den NFRs verwendeten Zahlen und Begriffe realistisch sind, und gibt Hinweise, welche überprüft werden sollten.

2.2.2.2.9 UC13 – NFRs verfeinern und referenzieren

Der Aktor überprüft die erstellten NFRs und überarbeitet/verfeinert sie wenn nötig.

2.2.2.2.10 UC 14 – Export Data

Ein Aktor möchte die erstellten NFRs und mögliche Zusatzoptionen wie Kommentare oder den Status des SMART-Kriteriums exportieren. Er hat die Möglichkeit zwischen verschiedenen Exportvarianten.

2.2.2.2.10.1 <<extends>> UC15 – AD-REPO- oder EEPPI-Import/-Export

Der Aktor exportiert die NFRs manuell in das AD-REPO- oder EEPPI-Tool oder importiert sie von dort.

2.2.2.2.10.2 <<extends>> UC16 – Generiere Report (human readable)

Der Aktor exportiert die NFRs als PDF oder Word Report.

2.2.2.2.10.3 <<extends>> UC17 – Generiere Report (machine readable)

Der Aktor exportiert die NFRs im XML-Format.

2.2.2.2.10.4 <<extends>> UC18 – NFRs als Task nach Task-Management-Tool exportieren

Der Aktor exportiert die NFRs in ein Projekt- und Issue-Management-Tool.

2.2.2.2.11 UC19 – NFRs kommentieren / akzeptieren / ablehnen (R-Kriterium)

Der Aktor verifiziert ein Set von NFRs auf deren Erfüllung des SMART-Kriteriums. Er akzeptiert oder lehnt jedes einzelne NFR ab und fügt wo nötig Kommentare hinzu, damit anschliessend der Architecture Synthesizer oder Analyst die NFR überarbeiten kann.

2.2.3 Nichtfunktionale Anforderungen

Die nachfolgende Tabelle enthält alle nichtfunktionalen Anforderungen. Sie wurden ebenfalls mit unserem BA-Betreuer diskutiert und zusammengefasst.

Nr.	**Category**	Priorität **[1 = required] [2 = nice to have] [3 = optional]**	Beschreibung
NFR1	Maintainability	1	Das Frontend und das Backend sollen zwei voneinander soweit unabhängige Komponenten sein, dass diese eigenständig verändert, weiterentwickelt und getestet werden können.
NFR2	Compatibility	1	Die komplette Funktionalität von QUALI-T muss in den beiden Browsern Firefox (v35.0.1) und Chrome (v43.0.2) den Benutzern vollumfänglich zur Verfügung stehen. Unterstützung von IE11 und Safari sind wünschenswert, aber nicht zwingend erforderlich, falls Mehraufwand von über 5h entsteht.
NFR3	Security	1	Zur Authentifizierung wird eine User ID mit einem Passwort verwendet. Diese User-Passwörter müssen mindestens acht Zeichen lang sein, ein Sonderzeichen und Gross-/Kleinschreibung enthalten.
NFR4	Usability	1	Innerhalb der QUALI-T-Applikation muss der Zugriff auf einzelne Funktionen über ein Rollenkonzept gesichert sein. Es sollen alle fünf Hauptrollen aus Kapitel 2.2.2.1 abgebildet sein. Wer Zugriff auf welche Funktionen hat, lässt sich aus der „Abbildung 6 Use-Case-Diagramm für QUALI-T“ ableiten. Zusätzlich gibt es eine Admin-Rolle, welche uneingeschränkte Berechtigung auf alle Funktionen hat.
NFR5	Performance efficiency	1	Alle Funktionen im Web GUI sollten innerhalb weniger als 1s ausführbar sein, unter Vernachlässigung des Netzwerks (beispielsweise beim lokalen Betrieb).
NFR6	Performance efficiency	1	QUALI-T muss mindestens 10 gleichzeitig angemeldete User auf dem Web Client unterstützen.
NFR7	Maintainability	1	Das Endprodukt muss ein lauffähiger Prototyp sein, der noch keine Produktivqualität hat. Der Zusatzaufwand zum Erreichen von Produktivqualität darf den Aufwand der BA jedoch nicht übersteigen.
NFR8	Usability	1	Nach einer maximalen Einarbeitungszeit von 3h ist der User in der Lage, neue Quality Attributes, Catalogs und Projects zu erstellen. Ausserdem versteht er den Zusammenhang und den Ablauf zwischen diesen drei Programm Modellen.
NFR9	Compatibility	1	Das Backend besitzt eine RESTful HTTP API für Backend-Integration aus jedem beliebigen Tool.

NFR10	Maintainability	2	Die in der Realisierung (der QUALI-T Software) verwendeten Tools und Frameworks sind vorzugsweise mit folgenden Lizenzen veröffentlicht: Eclipse, Apache License 2, Massachusetts Institute of Technology (MIT) oder Berkeley Software Distribution (BSD). Tools und Frameworks, die mit General Public License (GPL) und Lesser General Public License (LGPL) lizenziert sind, sollten nicht verwendet werden. Die einzige Ausnahme ist das Hibernate Framework (LGPL). Dieses darf nach Absprache mit unserem BA-Betreuer weiterhin verwendet werden.
NFR11	Usability	2	Um auch mehrstufige Quality Attribute Trees abbilden zu können, sollen mindestens fünf Hierarchiestufen für die Kategorisierung der QAs möglich sein.
NFR12	Usability	2	Wenn QA Templates verändert werden, wird in der Datenbank eine neue Version erstellt. Für den Anwender ist jeweils nur die aktuellste Version über QUALI-T aufrufbar. Gibt es aber bereits Instanzen von dem QA Template oder es wird in einem Katalog verwendet, so referenzieren diese QA Templates weiterhin auf die ursprüngliche Version.
NFR13	Compatibility	2	REST- und URI-Konzept sind so erstellt, dass Links von und auf andere Applikationen möglich sind. Beispiele für andere Anwendungen sind ADMentor/ADRepo, ART und EEPPI. Es soll eine Web-GUI-Koexistenz möglich sein, jedoch kein Web-Mashup oder Ähnliches.
NFR14	Reliability, Security	3	Alle Datenbankoperationen sollen die ACID-Eigenschaften (Atomicity, Consistency, Integrity, Durability) erfüllen. Während ein Objekt bearbeitet wird, soll dieses in der Datenbank gelockt werden und für andere nicht veränderbar sein. Wird versucht, auf ein gelocktes Objekt zuzugreifen, erhält der Anwender einen Warnhinweis.
NFR15	Security	3	Die Kommunikation zwischen Client und Server Tier ist als vertraulich klassifiziert (kein Mitlesen durch Dritte) und verwendet eine kryptographische Verschlüsselung (kein Manipulieren der Daten).
NFR16	Portability	3	QUALI-T muss lauffähig mit allen Funktionen auf Heroku deploybar sein.

Tabelle 5 Nichtfunktionale Anforderungen

3 QUALI-T

Im nachfolgenden Kapitel erläutern wir unsere Designentscheidungen und Konzepte, welche wir anschliessend umgesetzt haben. Ausserdem ist ein Überblick über das System, inkl. Continuous Integration and Cloud Deployment, erklärt.

Zusätzlich enthält dieses Kapitel einen Einblick in die RESTful HTTP API von QUALI-T.

3.1 Konzepte und Designentscheidungen

In diesem Unterkapitel sind die Konzepte von QUALI-T sowie die zur Realisierung notwendigen Designentscheidungen dokumentiert. Bei den dokumentierten Designentscheidungen haben wir uns auf die wichtigsten eingeschränkt.

3.1.1 Rollenkonzept

Aus dem Use Case-Diagramm haben wir die folgenden Rollen abgeleitet, welche in QUALI-T verfügbar sind und den Users zugeteilt werden können. Ein User kann mehrere Rollen haben.
CC = Catalog Curator
AA = Architecture Analyst
AS = Architecture Synthesizer
AE = Architecture Evaluator
ITP = IT Project Manager

Diese fünf Rollen haben innerhalb des Tools die in dieser Matrix aufgeführten Berechtigungen. Übergreifend gibt es noch eine Admin-Rolle, welche die Berechtigung für alles hat.

Permission	CC	AA	AS	AE	ITP	Admin
CRUD Catalog	x					x
CRUD Category	x					x
CRUD QA Template	x					x
CRUD Project		x	x	x	x	x
CRUD QA Instance		x	x	x	x	x
CRUD Project Initiator		x			x	x
CRUD Quality Property		x				x
Export to JIRA		x	x	x	x	x
Run XML/PDF Reports		x	x	x	x	x
CRUD QUALI-T User						x

Tabelle 1 Berechtigungsmatrix

3.1.2 QA-Konzept

Wird ein neues QA Template erstellt, wird es automatisch einem sogenannten Standard-Catalog hinzugefügt. Der Standard-Catalog beinhaltet alle QA Templates. Wird ein neuer Catalog erstellt, stehen dem Anwender die QA Templates und Variable Values des Standard-Catalog zur Verfügung. Die beiden Abläufe sind in den untenstehenden Sequenzdiagrammen vereinfacht erläutert.

In Abbildung 7 wird das QA mit den folgenden Angaben erstellt:

QA-Text QUALI-T muss *%VARIABLE_ENUMNUMBER_0%%* verfügbar sein.
Variables *VARIABLE_ENUMNUMBER_0*
Variable Values 99,9 / 99,5 / 99

Variable Type ENUMNUMBER

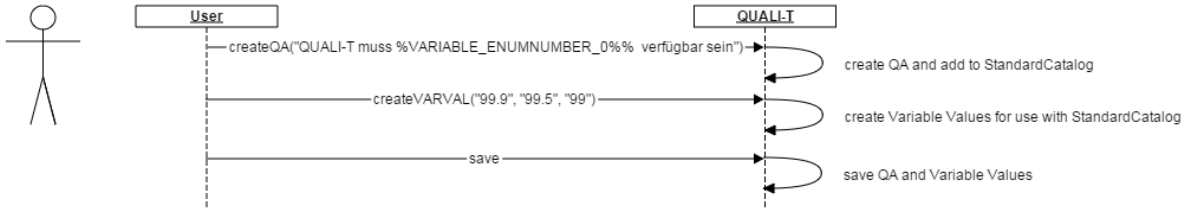


Abbildung 7 Sequenzdiagramm "Create Quality Attribute"

In Abbildung 8 wird ein Catalog mit dem Namen “CloudQAs“ erstellt. Anschliessend werden QAs aus dem Standard-Catalog hinzugefügt.

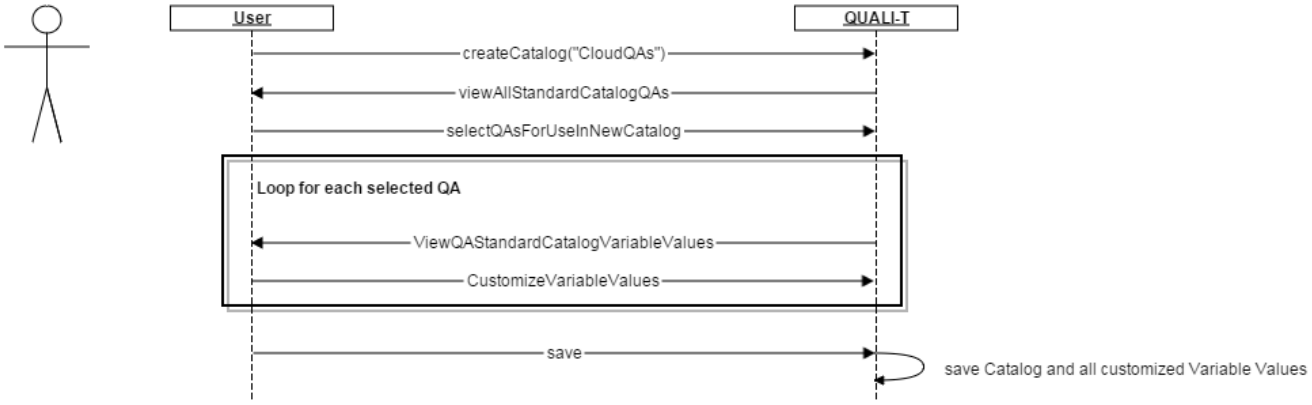


Abbildung 8 Sequenzdiagramm "Create Catalog"

Beim Erstellen eines Projektes gibt es zwei Möglichkeiten, die QAs hinzuzufügen:

- QAs aus Catalog: Es können beliebige QAs aus einem Catalog hinzugefügt werden. Nach der Auswahl müssen die Variablen gesetzt werden. Je nach Variablentyp verwendet man einen vorgegebenen Wert oder trägt eine Zeichenkette ein.
- Manuelle QAs: Zusätzlich können QAs direkt in einem Projekt erfasst werden. Es werden jedoch nur reine Text-QAs ohne Variablen unterstützt. Diese QAs stehen ausschliesslich in dem Projekt zur Verfügung und werden nicht als Template angelegt.

3.1.3 Variables

Eine Variable in einem QA kann als einen der folgenden drei Typen definiert sein:

Typ	Beschreibung
FREETEXT	Beliebige Zeichenfolge (alle Zeichen, inkl. Leerzeichen, erlaubt)
FREENUMBER	Beliebige Zahlen. Die Dezimalstelle kann durch einen Punkt getrennt werden.
ENUMTEXT / ENUMNUMBER	Vordefinierte Auswahlliste bestehend aus entweder FREETEXT oder FREENUMBER Werten. Können zudem mit eigenen Werten erweitert werden (extendable). Können einen Standardwert haben

Tabelle 6 Beschreibung der Variablentypen

Bei Auswahllisten besteht die Möglichkeit, einen Wert als Standardwert zu hinterlegen. Beim Ausfüllen einer QA Instance steht dann entweder ein Dropdown mit der Auswahlliste oder ein Eingabefeld zur Verfügung. Der Standardwert wird angezeigt, falls er vorhanden ist.

3.1.4 Löschverhalten

Nachfolgend wird beschrieben, welche Objekte im QUALI-T Frontend löschbar sind und welche Auswirkungen eine Löschung hat.

Frontend-Objekt	Auswirkung
Quality Attribute Template	Das Quality Attribute Template inkl. den dazugehörigen Variablen wird aus allen Catalogs gelöscht.*
Category	Es wird nur die QA Category gelöscht.
Catalog	Alle Catalog QAs werden gelöscht.*
Catalog QA	Das Catalog QA mit den zugehörigen Variablen wird gelöscht.*
Variable	Die Variable mit den zugehörigen Variable Values wird gelöscht.
Variable Value	Es wird nur der Variable Value gelöscht.
Project Initiator	Alle dem Project Initiator zugeordneten Projekte werden gelöscht.
Quality Property	Es wird nur das Quality Property gelöscht.
Project	Alle mit dem Projekt verknüpften QAs werden gelöscht.
Instance	Die QA Instance und die zugehörigen Variablen werden gelöscht.
User	Es wird nur der User gelöscht.

Tabelle 7 Löschverhalten der Frontend-Objekte

* Weitere Details sind im Kapitel 3.1.5 erklärt.

3.1.5 Versionierung

In der nachfolgenden Tabelle ist beschrieben, wie sich Operationen auf die QA Templates und die dazugehörigen Catalog QAs auswirken. Es ist somit sichergestellt, dass die Instances immer auf das Catalog QA referenzieren, aus dem sie erstellt wurden.

Operation	Auswirkung		Bemerkung
	QA Template	**Catalog QAs**	
QA Template Text editieren/Variable Type editieren	neue Version	neue Version	Die alten Versionen werden mit einem Löschattribut versehen.
QA Template Variable Values editieren	keine	keine	
QA Template löschen	„gelöscht“	„gelöscht“	Es wird beim QA Template und bei den Catalog QAs das Löschattribut gesetzt.
Catalog QA Variable Values editieren	keine	keine	
Catalog QA löschen	keine	„gelöscht“	Es wird beim Catalog QA das Löschattribut gesetzt.

Tabelle 8 Operationen auf Quality Attributes und deren Auswirkungen

Für die Realisation der Versionierung gibt es verschiedene Möglichkeiten. Wir haben diese gemäss der folgenden Designentscheidung evaluiert und umgesetzt.

Themengebiet	Versionierung	**Thema**	Design
Name	Version Control	**ID**	DES-DB-FW
Getroffene Entscheidung	Manuell implementierte Version Control		
Problemstellung	Wie soll die Versionierung von Objekten realisiert werden?		
Voraussetzung	Einzelne Objekte sollen in der Datenbank nicht löschbar sein und bei einer Änderung versioniert werden.		
Motivation	Um zu verhindern, dass Instances auf nicht mehr vorhandene oder nicht kompatible Catalog QAs referenzieren, sollen die Catalog QAs und QA Templates versioniert werden.		

Alternativen	• Database Version Control
	  ○ Vorteile
	  ▪ Versionierung aller Objekte möglich
	  ▪ Handling ausgelagert in Datenbank (QUALI-T muss sich nicht darum kümmern)
	  ○ Nachteile
	  ▪ Implementationsaufwand
	  ▪ Komplexer Zugriff auf ältere Versionen aus QUALI-T heraus		

Begründung	Wir benötigen nur die minimale Funktion eines Version Controlling, nämlich einzig die Versionsnummer. Dazu ein komplettes Version-Controlling-System auf Datenbankebene einzuführen, wäre überdimensioniert. Dazu kommt, dass beim Verwenden eines Database Version Controlling unsere Applikation an ein Datenbanksystem gebunden wäre und nicht einfach ausgetauscht werden könnte.		
Annahmen	Die Versionierung wird ausschliesslich für QA Templates und Catalog QAs benötigt und es ist kein Timestamp notwendig.		
Abgeleitete Anforderungen	Die Catalog QAs und QA-Template-Objekte werden mit einem „delete“-Attribute versehen, die QA-Template-Objekte zusätzlich mit den Attributen „versionNumber“ und „previousVersion“. Bei den Datenbankabfragen in QUALI-T auf alle Objekte einer Tabelle muss berücksichtigt werden, dass die Versionen mit dem „delete“-Vermerk nicht zurückgegeben werden.		
Verknüpfte Entscheidungen	keine		

Designentscheidung 1 Version Control

3.1.6 QA Categories

QAs können kategorisiert werden. Standardmässig werden die Kategorien des ISO25010 Standards gemäss Abbildung 9 verwendet. Diese bestehen aus zwei Hierarchiestufen mit acht Hauptkategorien und 31 Unterkategorien. Alternativ können auch eigene Kategorien und Unterkategorien erfasst werden [6]. Um auch komplexere Einteilungen wie z.B. bei den SEI Trees darzustellen, sind beliebig viele Hierarchiestufen unterstützt. In unserer BA wird jedoch nicht weiter auf diese Trees eingegangen, da wir uns für die Kategorisierung und Entscheidungshilfe des ISO-Standards entschieden haben [2].

Abbildung 9 ISO25010 Quality Properties [6]

3.1.7 Favorites

In QUALI-T kann jeder Benutzer Projects, an denen er aktuell arbeitet oder die er verfolgt, favorisieren. Ein favorisiertes Project erscheint danach als Quick Link in der rechten Navigationsleiste gemäss der folgenden Abbildung:

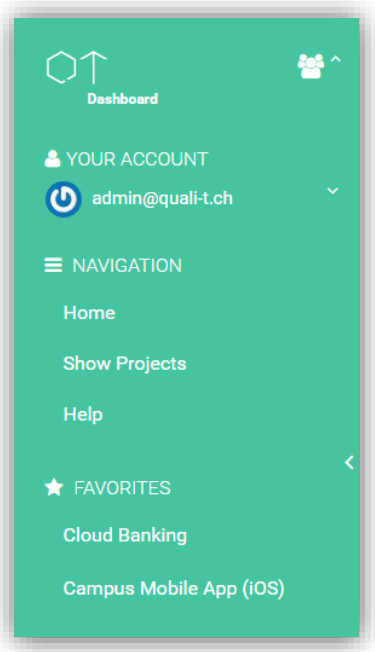


Abbildung 10 Navigationsleiste mit Favorites

Die Idee ist, dass anschliessend auf der Willkommensseite jeweils Updates zu den favorisierten Projects erscheinen. Dies kann zum Beispiel eine Information sein, dass ein neues QA hinzugefügt wurde oder dass der Status der QAs geändert wurde. Aus Zeitgründen konnten wir diese Funktion leider nicht mehr abschliessend implementieren. Das Favorisieren von Projects funktioniert jedoch einwandfrei.

3.1.8 Task Management

Es ist vorgesehen, dass einem Benutzer Tasks zu einem speziellen Project zugewiesen werden können. Der Ablauf ist wie folgt gedacht:

1. Architecture Analyst (AA) erstellt ein Project mit QAs
2. Er definiert den Architecture Synthesizer (AS), Architecture Evaluator (AE) und IT Project Manager (IPM) für dieses Project

3. Der AS und der AE erhalten pro QA einen Task, welche Eigenschaft sie prüfen müssen
   Beispiel AS: Task – Verify Specific and Measurable Property of QA ID 245
   Beispiel AE: Task – Verify Realistic Property of QA ID 245
   Beispiel IPM: Task – Verify Time-bound and Agreed Upon Property of QA ID 245
4. Nach erfolgreicher Prüfung können AS, AE und IPM den ihnen zugeordneten Property-Status für das QA ändern und wenn nötig einen Kommentar mit detaillierten Informationen dazu schreiben.

Alternativ können die Benutzer auch selber beliebige Tasks für sich erstellen. Aus Zeitgründen konnten wir diese Funktion nicht mehr implementieren. Im Backend existiert jedoch eine vorbereitete Klasse für das Task Management. Um die Funktion zu demonstrieren, existieren als Beispiel bereits einige initial erstellte Tasks wie in der folgenden Abbildung:

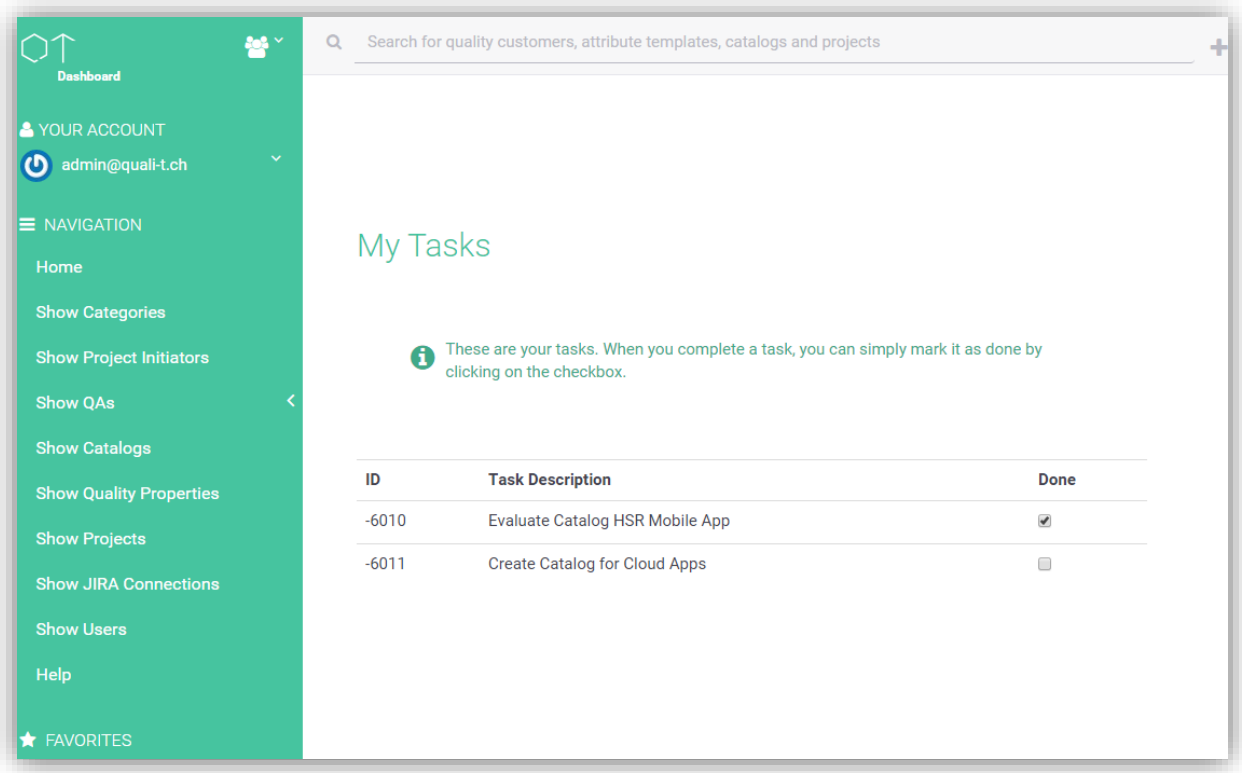


Abbildung 11 My Tasks in QUALI-T

3.1.9 Search

Die Daten in der Applikation sollen gesucht werden können. Wir haben hierfür keine komplizierten Suchalgorithmen entwickelt, da dies vom Ziel der Aufgabe abweicht. Deshalb haben wir uns für bewährte Suchtechniken, kombiniert mit Suchfunktionen der Datenbank, entschieden.

Die Suche wird auf dem Data Access Layer durchgeführt. Für Ressourcen, nach denen gesucht werden kann, muss im entsprechenden Data Access Object (z.B. für Project im ProjectDAO) eine Search-Methode implementiert werden.

Der Benutzer kann nach den folgenden Ressourcen suchen:

- Project
- Catalog
- Quality Attribute Templates
- Project Initiator

3.1.10 RESTful HTTP API

Im Request for Comments RFC 7231 wird das HTTP-Protokoll spezifiziert. Diese Spezifikation beinhaltet unter anderem HTTP-Verben und Status Codes [7]. Für QUALI-T haben wir diese Standards gebraucht. Die API von QUALI-T arbeitet mit JavaScript Object Notation (JSON). Sie erwartet also JSON-Objekte im Request und verwendet ebenfalls JSON in Responses.

3.1.10.1 Verben

In QUALI-T werden lediglich die HTTP-Verben DELETE, GET, POST und PUT verwendet. Eine detaillierte Erläuterung zu der Verwendung in QUALI-T ist in der nachfolgenden Tabelle enthalten.

Unsere Verwendung orientiert sich an der offiziellen Deklaration im IETF RFC 7231. Die möglichen Status Codes beschränken sich auf diejenigen, welche wir in unserer Applikation verwenden. Diese sind im Kapitel 3.1.10.2 genauer beschrieben.

Verb	Beschreibung / Anwendung
DELETE	Soll ein Objekt in QUALI-T gelöscht werden, so wird dies via DELETE-Verb gemacht. Da das Play Framework keinen Body für dieses Verb unterstützt (dieser wird ignoriert), wird die ID des zu löschenden Objekts direkt in der URL mitgegeben. Beispiel für das Löschen des Projekts mit der ID 1: DELETE http://localhost:9000/api/project/1
GET	Sämtliche Objekte werden über GET angefordert. Wird eine Liste von beispielsweise allen Projekten angefordert, werden keine Parameter verwendet: GET http://localhost:9000/api/project Wird hingegen nur ein spezielles Projekt abgefragt, wird die ID ebenfalls als Parameter übergeben: GET http://localhost:9000/api/project/1

POST	POST wird verwendet, um neue Objekte zu erstellen. Alle notwendigen Parameter werden im HTTP Body im JSON-Format übermittelt:
	POST http://localhost:9000/api/project
	HTTP Body:
	{
	"name": "Project 1",
	"projectInitiator": 3000,
	"qps": [
	{
	"id": 8000,
	"name": "S",
	"description": "Specific"
	}
	],
	"qas": [
	{
	"id": 5000,
	"description": " Das System soll _____ % verfügbar sein. "
	}
	],
	"catalog": 6001
	}

PUT	Wird ein Objekt editiert, so werden beim Übermitteln alle notwendigen Parameter im HTTP Body als JSON via PUT ins Backend geschickt. Wichtig ist hier, dass die ID im JSON mitgegeben wird und nicht in der URI. PUT http://localhost:9000/api/project { “id”:1001, "name": "Project editiert", "projectInitiator": 3001, "qps": [{ "id": 8000,

```
            "name": "S",
            "description": "Specific"
        }
    ],
    "qas": [
        {
            "id": 5000,
            "description": "<p>Das System soll _____ % verfügbar sein.</p>"
        }
    ],
    "catalog": 6001
}
```


Tabelle 9 Benutzung der HTTP-Verben in QUALI-T

3.1.10.2 Status Codes

Für das QUALI-T-Backend verwenden wir bereits bestehenden Status Codes. Eine allgemeine Erläuterung dazu ist in Tabelle 10 und Tabelle 11 zu finden. Informationen zu den QUALI-T Exceptions und den dazugehörigen Status Codes sind im Kapitel 3.1.10.3 dokumentiert.

Status Codes Range	Beschreibung gemäss RFC 7231 [7]	Benutzung in QUALI-T
1xx	Informational – Request received, continuing process	--
2xx	Success – The action was successfully received, understood, and accepted	200 OK with Object 204 OK without Content
3xx	Redirection – Further action must be taken in order to complete the request	--
4xx	Client Error – The request contains bad syntax or cannot be fulfilled	400 Bad Request 401 Unauthorized Access 403 Access Denied 404 Not Found 422 Unprocessable Entity
5xx	Server Error – The server failed to fulfill an apparently valid request	Diese Exceptions werden nicht explizit in QUALI-T geworfen. Ist aber beispielsweise der Server nicht erreichbar oder antwortet nicht, wird eine solche Fehlermeldung generiert.

Tabelle 10 Allgemeine Übersicht der HTTP Status Codes

Status Codes	Beschreibung	Benutzung in QUALI-T
200	OK with Object	Jeder Request, der erfolgreich verarbeitet wurde und eine Response mit Inhalt (JSON-Objekt) liefert, wird mit diesem Status Code beantwortet.
204	OK without Content	Requests, die erfolgreich verarbeitet werden, aber kein Objekt in der Response enthalten (z.B. bei Löschvorgängen), werden mit 202 beantwortet.
400	Bad Request	Wird ein Request semantisch und syntaktisch korrekt gestellt, das angeforderte Objekt existiert jedoch nicht in der DB, so wird eine 400 Response verschickt. Dieser Status Code wird zusätzlich für alle

		undefinierten Errors verwendet.
401	Unauthorized Access	Wird versucht, auf eine Backend URL zuzugreifen, für die der eingeloggte User keine Berechtigung hat, so wird dieser Status Code als Response übermittelt. Das Gleiche gilt für den Fall, dass versucht wird, ohne Authentifizierung auf eine gesicherte Ressource zuzugreifen.
403	Access Denied	Wird im Frontend eine Seite aufgerufen, auf die der eingeloggte User keinen Zugriff hat, so wird eine Access Denied Message angezeigt. Vom Server wird in der HTTP Response jedoch ein „304 Not Modified“ Status Code verschickt, welchen das Frontend aber als 403 interpretiert.
404	Not Found	Wird eine URL aufgerufen, die im Backend nicht existiert, wird dies mittels 404 Status Code beantwortet. Dieser Code wird direkt vom Play Framework versendet.
422	Unprocessable Entity	Enthält ein Request zu wenige oder semantisch falsch formulierte Parameter, die URL ist aber korrekt, so wird dieser Code in der Response verwendet.[5] Dies trifft ebenfalls zu, wenn versucht wird, ein Objekt zu erstellen, welches bereits existiert.

Tabelle 12 Übersicht der in QUALI-T verwendeten Status Codes

3.1.10.3 QUALI-T Exceptions

Wir haben uns dazu entschieden, das Error Handling im Backend mit Exceptions (eigene Exceptions) zu realisieren. Die Exceptions werden im Logic Layer geworfen, entweder direkt von den Logik-Klassen oder von den Database Access Objects (DAOs). Abgefangen werden die Exceptions in den jeweiligen Controllern, welche die Exception Message mit dem dazugehörigen Status Code an das Frontend schicken. Die Exceptions werden in den jeweiligen Controllern über ein implementiertes Functional Interface[6] abgefangen und mit dem dazugehörigen Status Code als Rückgabewert an das Frontend übergeben.

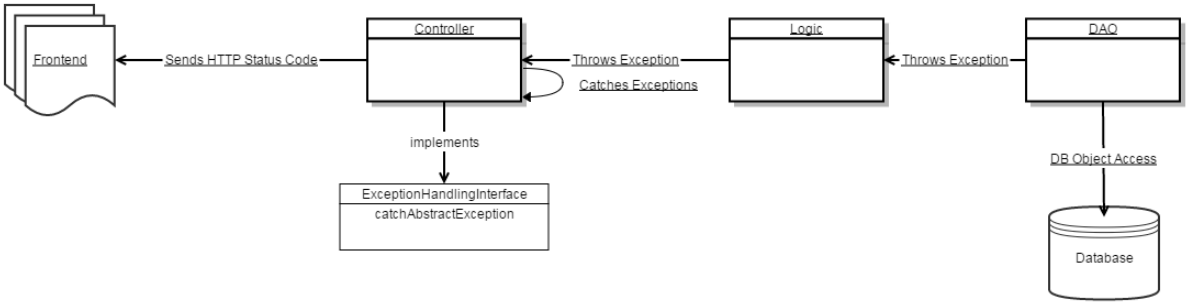


Abbildung 13 Exception Handling

[5] Dieser Code ist gemäss Definition für WebDAV Responses vorgesehen. Er wird aber vermehrt für syntaktisch korrekte, aber semantisch falsch formulierte Requests verwendet. [17]

[6] https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html, zuletzt aufgerufen am 11.06.2015

Diese Tabelle zeigt, welche Exceptions auftreten können und welche Status Codes sie auslösen.

Exception	Bedeutung	Status Code
AbstractException	Ist die Superklasse aller nachfolgenden Exceptions und dient dem vereinfachten Abfangen der Exceptions. Die AbstractException wird jedoch nie selber geworfen. Enthält zusätzlich zur Message den Status Code.	
CouldNotConvertException	Beim Import oder Export von Daten in QUALI-T ist ein Server Fehler aufgetreten. Der Server konnte die Daten nicht erfolgreich Konvertieren.	500
EntityAlreadyExistsException	Es wird versucht, ein Objekt zu erzeugen, das bereits existiert. Dies kann vorkommen, wenn ein Attribut auf *unique* gesetzt ist (z.B. ProjectInitiator name oder Project name).	422
EntityCanNotBeDeleted	Es wird versucht, eine Entity zu löschen, die nicht gelöscht werden darf.	403
EntityCanNotBeUpdated	Es wird versucht, eine Entity zu editieren, die nicht editiert werden darf.	403
EntityNotCreatedException	Das Objekt konnte aus unbekanntem Grund nicht erstellt werden.	400
EntityNotFoundException	Ein Request verweist auf eine Objekt-ID, welche in der Datenbank nicht gefunden werden kann.	400
InvalidConnectionParameter	Die Verbindungsparameter zu einem externen System sind nicht korrekt (z.B. JIRA-Verbindung).	422
MissingParameterException	Ein Parameter für das Erstellen des gewünschten Objekts fehlt, z.B. der ProjectInitiator name beim Erstellen eines neuen ProjectInitiators.	422
PasswordNotMatchException	Falls das eingegebene Passwort nicht mit dem in QUALI-T hinterlegten übereinstimmt, wird diese Exception geworfen.	422
PasswordsNotMatchException	Das eingegebene Passwort ist nicht korrekt oder kann im aktuellen Kontext nicht verwendet werden.	422

Tabelle 11 Custom QUALI-T Exceptions

3.1.11 Statistiken

Qualitätsattribute, welche in QUALI-T erfasst werden, sind in der Datenbank abgespeichert. So lassen sich Statistiken berechnen, die dem Benutzer bei der Eingabe von neuen Informationen helfen können. In der initialen Version der Webapplikation werden Statistiken über Variablenwerte berechnet. Statistiken werden für alle Variablentypen ausser FREETEXT geführt, denn dort sind Statistiken nicht sinnvoll. Statistiken werden statisch in den Variablen (QAVar) gespeichert und bei Änderungen an Instanzen (im Projekt) angepasst. So ist zu jeder Zeit eine aktuelle Statistik gewährleistet.

Denkbar wäre folgendes Szenario:

1. Der Benutzer (IT Project Manager) möchte ein neues Project anlegen und wählt hierfür den bereits vorhandenen „iOS App Catalog“ aus.

2. Das Project ist erstellt und enthält alle Quality Properties, welche im Catalog definiert wurden.
3. Der Benutzer startet mit dem Ausfüllen der Variablen.
4. Der Benutzer validiert das Project.
5. Eingegebene Variablenwerte werden mit statistischen Werten verglichen. Falls Warnungen (Abweichungen von statistischen Werten) existieren, werden diese angezeigt.

3.1.11.1 Meistbenutzter Wert

Beim meistbenutzten Wert wird die Anzahl Vorkommnisse eines Wertes gezählt. Der höchste Wert wird hierbei in der Statistik gespeichert.

3.1.11.2 Durchschnittswert

Beim Durchschnittswert sind nur Zahlenwerte (FREENUMBER, ENUMNUMBER) betroffen. Es wird der arithmetische Durchschnitt der jeweiligen ausgefüllten Variablenwerte der gleichen Variable berechnet.

3.1.12 Fuzziness Detector API

Um spezifizierte Quality Attributes zu überprüfen, haben wir die Fuzziness Detector API entwickelt. In diesem Kapitel erläutern wir, den Nutzen der Fuzziness Detector API. Zudem wird eine Implementation der API „Fuzziness Detectors“ genannt.

3.1.12.1 Anforderung

Fuzziness Detectors werden beim Spezifizieren der Qualitätsattribute innerhalb eines Projekts verwendet. Sie geben dem Benutzer Feedback, welche Qualitätsattribute eventuell definierte Regeln verletzen und deshalb detektiert wurden. Ein guter Fuzziness Detector assistiert den Benutzer und hilft mit Verbesserungsvorschlägen.

Der Hauptgrund, warum eine API entwickelt wurde, ist, dass die Applikation mit Detektoren beliebig erweitert werden soll. QUALI-T unterstützt also eine einfach Einbindung von Fuzziness Detectors. In Anbetracht dessen, dass die Benutzer von QUALI-T in den meisten Fällen einen technischen Hintergrund haben, erwarten wir, dass sie ihre eigenen Fuzziness Detectors entwickeln und verwenden werden.

3.1.12.2 Design

In der ersten Version bietet die Fuzziness Detector API eine Methode für die Validierung an. Die Implementation dieser Methode soll eine Zeichenkette, die validiert werden soll, übergeben. Der Rückgabewert der Methode ist eine Liste von Verbesserungsvorschlägen).

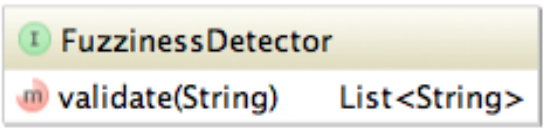


Abbildung 14 Service Provider Interface FuzzinessDetector

3.1.12.2.1 Automatische Injektion von Fuzziness Detectors mittels ServiceLoader

Um die Einbindung von selbstimplementierten Detektoren zu vereinfachen haben wir die ServiceLoader[7] Funktionalität aus dem Java Development Kit (ab Java Version 1.6) benutzt. Beim

[7] Die Klasse ServiceLoader ist seit Java 1.6 implementiert und unter https://docs.oracle.com/javase/6/docs/api/java/util/ServiceLoader.html dokumentiert. (zuletzt aufgerufen am 04.06.2015)

Laden eines Services (implementiert FuzzinessDetector Interface) kann man den Typ spezifizieren. Aufgrund dieser Spezifikation werden die Detektoren (welche das Interface implementieren) gefunden und deren validate-Methode kann aufgerufen werden.

3.1.12.2.2 META-INF/services

Für das richtige Funktionieren des ServiceLoaders braucht jeder Detektor eine Datei im META-INF/services Ordner. Die Datei hat den Fully Qualified Name (FQN) des implementierten Service Provider Interface mit dem FQN zur Implementation des SPI.

```
ch.qualit.fuzziness.detector.blacklist.BlacklistDetector
```

Code 1 Fully Qualified Name (FQN) des Blacklist Detector

3.1.12.3 Built-in Detector „Blacklist Detector“

Für die Bachelorarbeit haben wir den Blacklist Detector entwickelt. Dieser enthält eine Liste von Wörtern, welche im Text nicht enthalten sein sollten. Zu jedem Wort gibt es eine Liste von Verbesserungsvorschlägen.

3.1.12.3.1 Beispiel

Wenn die untenstehende Zeichenkette validiert wird, sollte der Detektor aktiv werden.

```
The System has a good availability.
```

Code 2 Fully Qualified Name (FQN) des Blacklist Detector

Das Wort “good” ist in der Blacklist enthalten und wird detektiert, weil es nicht messbar ist.

3.2 Tool- und Framework-Evaluationen

Nachfolgend werden Designentscheidungen für verwendete Tools und Frameworks beschrieben. Diese Entscheidungen wurden mittels dem IBM UMF Template for Decision Log[8] dokumentiert.

3.2.1 Client-Framework-Evaluation

Für die Client-Applikation haben wir eine Evaluation von bestehenden Frameworks durchgeführt. Die entscheidenden Kriterien sind in der Tabelle unten ersichtlich. Für die Bewertung der Kriterien haben wir in der Elaboration-Phase ein Hands-on mit den zwei Frameworks durchgeführt. Zusätzlich zu unserer eigenen Erfahrung mit AngularJS und React haben wir uns in diversen Blogs [8] [9] [10] erkundigt.

Bei der Auswahl der zu evaluierenden Frameworks haben wir uns beschränkt auf AngularJS (v1.3.14)[9] und React (v0.12.2)[10], da andere Frameworks bereits im Rahmen der Bachelorarbeit EEPPI[11] evaluiert wurden. Die Anforderungen für QUALI-T sind mit denen von EEPPI vergleichbar.

Wir haben uns für AngularJS entschieden, da wir das Framework bei der Evaluation am besten bewerten konnten und es alle Kriterien mit der höchsten Wichtigkeit erfüllt.

[8] Das Template ist verfügbar unter https://files.ifi.uzh.ch/rerg/amadeus/teaching/courses/it_architekturen_hs10/Solution_Design_I.day.pdf (zuletzt aufgerufen am 02.03.2015)
[9] https://angularjs.org (zuletzt aufgerufen am 02.03.2015)
[10] https://facebook.github.io/react (zuletzt aufgerufen am 02.03.2015)
[11] http://eprints.hsr.ch/393 (zuletzt aufgerufen am 11.06.2015)

Kriterium	Wichtigkeit 1=sehr wichtig 2=wichtig 3=vernachlässigbar	AngularJS	React
Community	2	✓	O
Ease Of Use	2	O	O
Fits in current technology stack	1	✓	X
Fits in project requirements	1	✓	O
Performance	2	O	✓
Stability	1	✓	X (nicht stabil v0.12.2)
Support	2	✓	O
Testability	2	✓	✓
Tool Support (IDE, Command Line Tools, Validators, etc.)	2	✓	O
Use of existing libraries	3	✓	X

Tabelle 12 Bewertungskriterien für AngularJS und React

Legende für die Bewertung
x = Kriterium ist ungenügend / nicht erfüllt
o = Kriterium ist grösstenteils erfüllt
✓ **= Kriterium ist erfüllt**

Themengebiet	Framework	**Thema**	Technologie
Name	Auswahl Client Framework	**ID**	TEC-CLIENT-FW
Getroffene Entscheidung	AngularJS		
Problemstellung	Welches Client Framework soll gebraucht werden?		
Voraussetzung	Es wird eine Client-zentrierte Webapplikation entwickelt. Das UI-Rendering passiert beim Client im Browser statt auf dem Server.		
Motivation	Diese Entscheidung beeinflusst die gesamte Architektur der Applikation. Für die Entscheidung ist vor allem die Heterogenität mit bestehenden Applikationsarchitekturen (EEPPI) in den Vordergrund zu stellen. Ausserdem sollen etablierte Patterns (MVC mit Data-Binding) nicht manuell von den Entwicklern implementiert werden, sondern bereits existierende Komponenten verwendet werden können.		

Alternativen	• React
	  ○ Vorteile
	  ▪ Performant bei vielen GUI-Elementen
	  ▪ Wird im Gegensatz zu AngularJS (2.0) sehr wahrscheinlich keine grossen Änderungen in naher Zukunft erleben → Know-how bleibt länger erhalten
	  ○ Nachteile
	  ▪ Kein stable release (Version < 1.x)
	  ▪ Auftraggeber hat kein Know-how, was für die Weiterentwicklung gebraucht wird		

Begründung	AngularJS wurde bereits für ein Projekt im HSR Institut IFS angewendet und ist im Gegensatz zu React schon seit langer Zeit stabil und mittlerweile state of the art.		
Annahmen	Keine		
Abgeleitete Anforderungen	Der Server bietet eine Service API (RESTful HTTP) an, sodass die AngularJS-Applikation mit Daten arbeiten kann (read and write).		
Verknüpfte Entscheidungen	Keine		

Designentscheidung 2 Client-Framework-Evaluation

3.2.2 IT-Automation-Evaluation

Um die virtuellen Maschinen, welche für die Entwicklung genutzt werden, mit den nötigen Tools wie JDK, Activator und weiterer Software auszustatten, wollen wir eine IT-Automation-Software benutzen. Somit muss die virtuelle Maschine nur noch gestartet werden und erfordert keine weiteren Installationsschritte.

Für die Evaluation haben wir zwei Softwareprodukte, die stark mit Vagrant zusammenarbeiten, ausgewählt.

- Puppet
- Ansible

Themengebiet	Environment	**Thema**	Software
Name	Auswahl IT Automation	**ID**	ENV-ITAUT-SW
Getroffene Entscheidung	Puppet		
Problemstellung	Welche Software soll für die IT Automation verwendet werden?		
Voraussetzung	Automatische Installation von Linux-Packages und Support von Vagrant		
Motivation	Diese Entscheidung ist für die Zeit während der Bachelorarbeit nicht weiter wichtig, denn nach dem initialen Setup werden mit grosser Wahrscheinlichkeit wenige Änderungen durchgeführt. Für die spätere Weiterentwicklung ist die Entscheidung jedoch wichtig. Mit dem richtig ausgewählten Produkt kann QUALI-T in Zukunft leichter erweitert werden.		

Alternativen	• Ansible
	  ○ Vorteile
	  ▪ Software muss auf dem Zielrechner nicht installiert sein
	  ▪ Skaliert sehr gut mit mehr Hosts, die verwaltet werden müssen
	  ○ Nachteil
	  ▪ Keine Unterstützung von Windows		

Begründung	Wir haben uns für Puppet entschieden, da Ansible ihre Software nicht für Windows zur Verfügung stellt. Somit können Entwickler nicht auf Windows die virtuelle Maschine aufsetzen.		
Annahmen	Keine		
Abgeleitete Anforderungen	Keine		
Verknüpfte Entscheidungen	Keine		

Designentscheidung 3 IT-Automation-Evaluation

Wir haben das Environment mit Puppet und Vagrant aufgebaut. Jedoch hatten wir diverse Schwierigkeiten. So war die Performance auf unseren Notebooks mit 8GB RAM viel zu schlecht. Nachdem die virtuelle Maschine gestartet war und wir mit der Programmierarbeit begonnen hatten, gab es immer wieder unerklärliche Abstürze. Trotz intensiver Troubleshooting konnten wir die Probleme nicht in angemessener Zeit lösen. Deshalb entschieden wir uns, die Entwicklungsumgebung lokal auf unseren Notebooks zu installieren. Unsere Erfahrungen sowie die Installationsanleitung haben wir im Anhang in Kapitel **Error! Reference source not found.** ausführlich dokumentiert.

3.2.3 Datenbank-Evaluation

Für das Backend wird zwingend eine Datenbank benötigt, um die Daten (z.B. Kundendaten, NFRs, Templates) zu persistieren.

Zur Auswahl standen die beiden Open-Source-Anwendungen

- MySQL
- PostgreSQL

Themengebiet	Datenbank	**Thema**	Technologie
Name	Auswahl Backend-Datenbank	**ID**	TEC-SERVER-DB
Getroffene Entscheidung	PostgreSQL		
Problemstellung	Welche Datenbank soll für das Persistieren verwendet werden?		
Voraussetzung	Es soll eine RDBMS-Datenbank benutzt werden, welche auf einer Open-Source-Lizenz basiert. Die Schnittstelle muss möglichst allgemein programmiert werden, um später ohne grösseren Aufwand auf ein anderes DB-System wechseln zu können. Die Datenbank muss eine Java API unterstützen.		
Motivation	Die Entscheidung ist wichtig, da die Datenbank eine der ersten und die wichtigste Schnittstelle ist, die angebunden wird und bereits im Prototyp vorhanden sein muss.		

Alternativen	• MySQL
	  ○ Vorteile
	  ▪ Viele Security Features
	  ▪ Gute Community
	  ▪ Unterstützung durch die meisten Hosting Provider
	  ▪ Einfache Installation und Verwaltung durch GUIs
	  ○ Nachteile
	  ▪ Development-Prozess seit Kauf durch Oracle stark rückläufig
	  ▪ Eingeschränkte Funktionen im Vergleich mit anderen RDBMS (Transaktionen, Auditing usw.)		

Begründung	PostgreSQL wurde bereits für vorgängige Projekte (z.B. EEPPI) im HSR Institut IFS angewendet und wir haben damit in den Datenbankvorlesungen positive Erfahrung gemacht. Ausserdem werden viele komplizierteren Datentypen unterstützt. Eine spätere Migration auf ein proprietäres Produkt von z.B. Oracle oder Microsoft ist ohne grossen Aufwand möglich.		
Annahmen	Keine		
Abgeleitete Anforderungen	Auf die Datenbank wird mittels JDBC Treiber von PostgreSQL[12] zugegriffen.		
Verknüpfte Entscheidungen	Keine		

Designentscheidung 4 Datenbank-Evaluation [11]

3.2.4 Framework für Dependency Injection

Im Backend soll Dependency Injection benutzt werden. Hierfür gibt es verschiedene Frameworks. Wir haben deshalb folgende Frameworks für die spätere Integration evaluiert:

- Google Juice
- Spring Framework (spring-context)

[12] https://jdbc.postgresql.org (zuletzt aufgerufen am 10.06.2016)

Themengebiet	Frameworks	**Thema**	Technologie
Name	Auswahl Dependency Injection Framework	**ID**	TEC-SERVER-DI
Getroffene Entscheidung	Google Guice		
Problemstellung	Welches Framework soll für Dependency Injection benutzt werden?		
Voraussetzung	Es wird ein Framework für Dependency Injection benötigt, welches sich einfach in die Backend-Applikation (Play Framework) integrieren lässt.		
Motivation	Die Entscheidung ist nicht sehr wichtig. Wichtiger ist, dass ein Framework gefunden wird.		

Alternativen	• Spring Framework
	  ○ Vorteile
	  ▪ Harmoniert mit anderen Komponenten aus dem Spring Framework
	  ▪ State of the Art
	  ○ Nachteil
	  ▪ Probleme bei Integration ins Play Framework (Details im Kapitel 3.3.3.4.1)		

Begründung	Wir haben uns für Google Guice entschieden, da das Framework sich in Play 2.3 erfolgreich integrieren lässt.		
Annahmen	Keine		
Abgeleitete Anforderungen	Keine		
Verknüpfte Entscheidungen	Keine		

Designentscheidung 5 Dependency Injection Framework Evaluation

3.2.5 Projektmanagement-Tool-Evaluation

Damit die Planung der einzelnen NFR Instances sichergestellt werden kann, wird ein Export in ein Projektmanagement-Tool zur Verfügung gestellt. Zur Auswahl stehen folgende Produkte:

- Trello
- JIRA

Themengebiet	Schnittstellen	**Thema**	Technologie
Name	Auswahl Schnittstelle Projektmanagement-Tool	**ID**	TEC-SERVER-PMT
Getroffene Entscheidung	JIRA		
Problemstellung	Zu welchem Projektmanagement-Tool soll eine Schnittstelle implementiert werden?		
Voraussetzung	Es wird ein Projektmanagement-Tool benötigt, in das die NFRs mittels RESTful HTTP API als Tasks importiert werden können. Dort werden die NFRs Tasks geplant und den zuständigen Personen zugewiesen.		
Motivation	Die Entscheidung ist wichtig, da die Qualitätseigenschaften A und T nicht via QUALI-T, sondern via Projektmanagement-Tool sichergestellt werden.		

Alternativen	• Trello
	  ○ Vorteile
	  ▪ Einfache und intuitive Bedienung
	  ▪ Apps für Android und iOS verfügbar
	  ▪ Rechtevergabe ist simpel zu managen (via E-Mail-Account direkt auf Listen oder Projekte)
	  ○ Nachteile
	  ▪ Keine Historie (Änderungsverfolgung) der einzelnen Aufgaben
	  ▪ Bei grossen Projekten mit vielen Aufgaben und Listen verliert man schnell den Überblick im Dashboard		

Begründung	Wir haben uns für das JIRA-Projektmanagement-Tool entschieden, da wir beide bereits Erfahrung damit gemacht haben. Zudem haben wir dank unserer BA Zugriff auf eine aktualisierte Version, da wir unser Issue Management darin realisiert haben. Aus unserer Sicht ist JIRA aktuell auch die bekanntere und besser etablierte Lösung für Softwareentwicklungsprojekte als Trello.		
Annahmen	Keine		
Abgeleitete Anforderungen	Der Netzwerkzugriff von der Applikationsumgebung auf den JIRA Server muss sichergestellt werden.		
Verknüpfte Entscheidungen	Keine		

Designentscheidung 6 Projektmanagement-Tool-Schnittstelle [12]

3.3 Design und Implementation

In diesem Abschnitt wird auf das Design und die Implementation von QUALI-T eingegangen. Dazu gehört beispielweise die Architektur- und Systembeschreibung. Anschliessend ist das Kapitel weiter aufgespaltet in einen Frontend- und einen Backendabschnitt gefolgt von Kapiteln, welche beide Applikationsteile betreffen.

3.3.1 Architektur

Wir haben die Applikation in zwei eigenständige Applikationen (Frontend und Backend) aufgeteilt und QUALI-T als eine 2-Tier-Architektur entwickelt.
Während im Client-Tier nur der Presentation Layer implementiert ist, ist der Server-Tier für den Application Logic und Resource Layer zuständig. In der nachfolgenden Abbildung sind diese drei Layer mit den darin verwendeten Frameworks sowie den dazugehörigen Komponenten zu sehen.

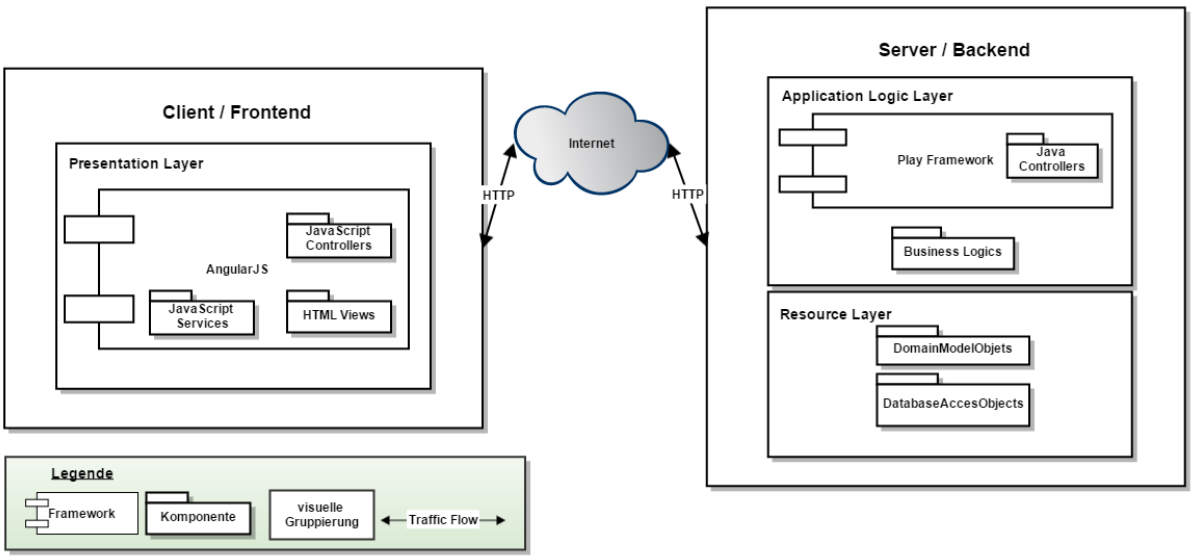


Abbildung 15 Verteilung der logischen Layers auf die physischen Tiers

Wie diese Komponenten in QUALI-T miteinander verknüpft sind zeigt die Abbildung 16 Flow zwischen den Komponenten der logischen Layer. Es wird ein kompletter Aufruf von einem Client bis hin zur Speicherung in der Datenbank dargestellt. Dabei ist zu erkennen, wie die einzelnen Komponenten miteinander agieren. Die Response ist nicht dargestellt, sie geht jedoch genau den gleichen Weg zurück. Auf die einzelnen Frameworks und Komponenten wird in den nachfolgenden Kapiteln 3.3.2 und 3.3.3 detaillierter eingegangen.

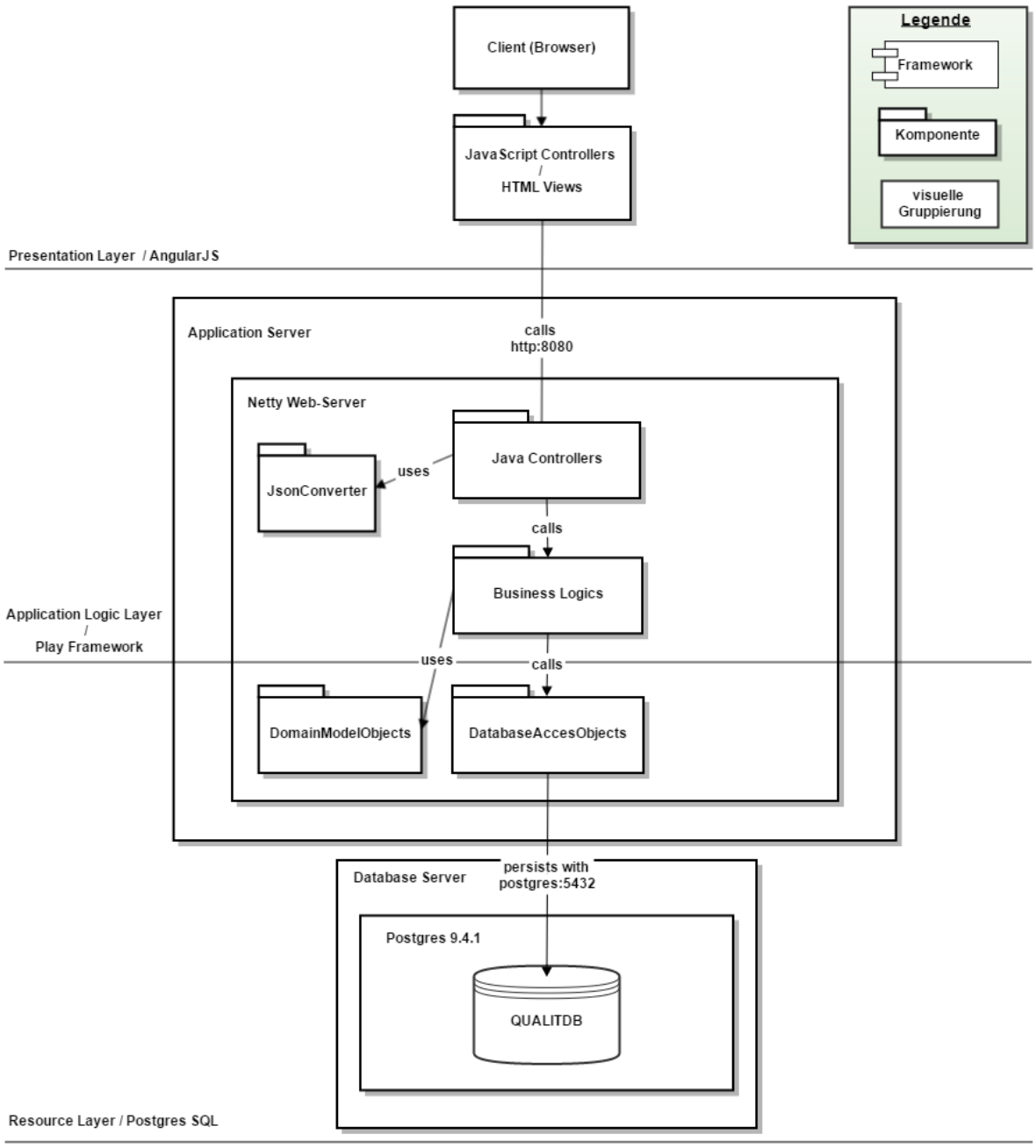


Abbildung 16 Flow zwischen den Komponenten der logischen Layer

3.3.1.1 Systemüberblick

Im nachfolgenden Diagramm haben wir das System mit den jeweiligen Komponenten und deren Abhängigkeiten visualisiert.

Für die Bachelorarbeit haben wir uns entschieden, Continuous Integration und Deployment in die Cloud zu verlagern. Hierfür haben wir Travis CI als Continuous Integration Service und Heroku als Cloud-Plattform ausgewählt.

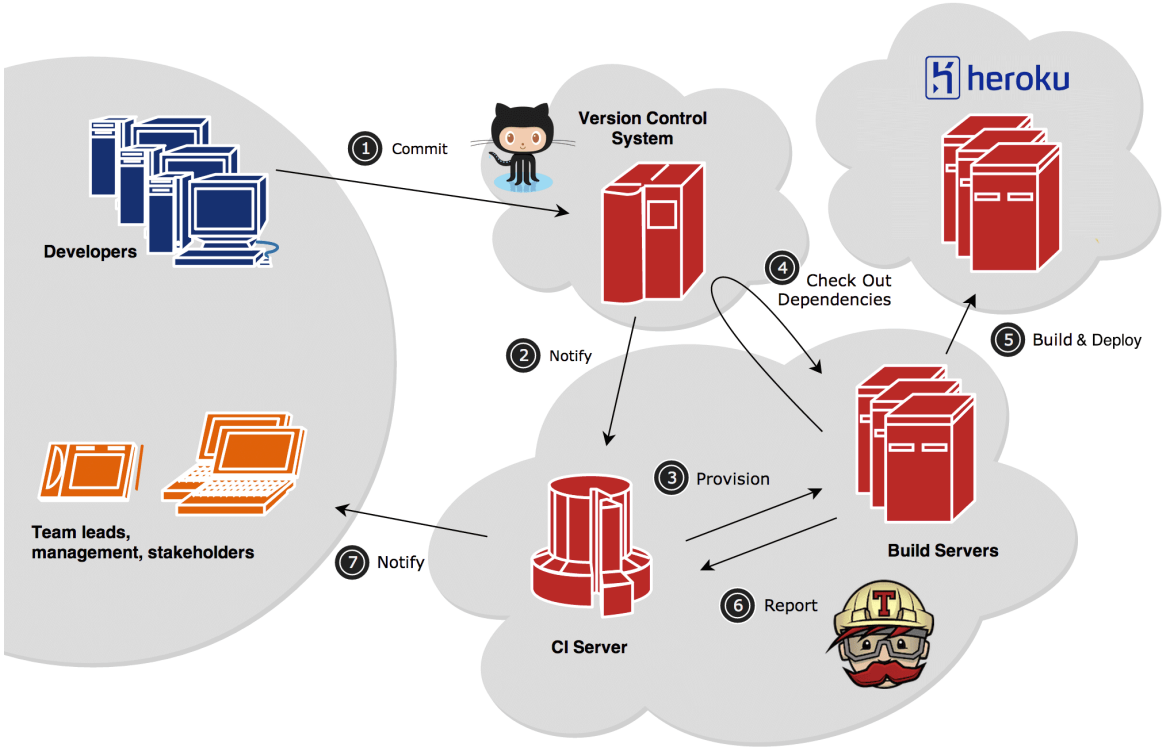


Abbildung 17 Systemübersicht mit Continuous Integration und Cloud Deployment
Quelle: Original Bild von http://decks.eric.pe/pantheon-ci/images/ci-architecture-pantheon.png überarbeitet

Bei jedem Git Commit & Push[13] (Schritt 1 – Commit) vom Entwickler zum zentralen Git Repository wird der Continuous Integration Server von Travis CI notifiziert (Schritt 2 – Notify). Dieser Server setzt gemäss Konfiguration (.travis.yml Datei) eine Maschine auf und installiert diese (Schritt 3 – Provision). Typischerweise enthält die Konfiguration Anweisungen für die gesamte Installation, inkl. Checkout des Github-Projekts (Schritt 4 – Check Out Dependencies) der Applikation, sowie für das Testing. Zusätzlich haben wir nach erfolgreichem Installieren und Testen Anweisungen für das Deployment in die Heroku Cloud konfiguriert (Schritt 5 – Build & Deploy). Im Travis CI Dashboard sind für alle Commits die von Travis erzeugten Reports ersichtlich (Schritt 6 – Report). Bei fehlgeschlagenen Reports wird jeweils der Entwickler, welcher den Schritt 1 ausgeführt hat und somit den Build zum Fehlschlagen gebracht hat, mit einer E-Mail notifiziert (Schritt 7 – Notify).

3.3.2 Frontend-Applikation

Die Frontend-Applikation braucht das AngularJS Framework (v1.3.14). In der Elaboration-Phase wurde dieses Framework evaluiert (siehe Kapitel 3.2.1) und ausgewählt. Die Applikation konsumiert die von der Backend-Applikation zur Verfügung gestellte API.

3.3.2.1 Architektur

Da AngularJS eine bewährte Architektur hat, haben wir diese Standardkomponenten[14] gebraucht:

[13] Weitere Informationen zu den GIT-Kommandos sind unter http://git-scm.com/docs/git-push verfügbar (zuletzt aufgerufen am 15.03.15)
[14] Die komplette AngularJS Guide ist unter https://docs.angularjs.org/guide verfügbar (zuletzt aufgerufen am 05.06.2015)

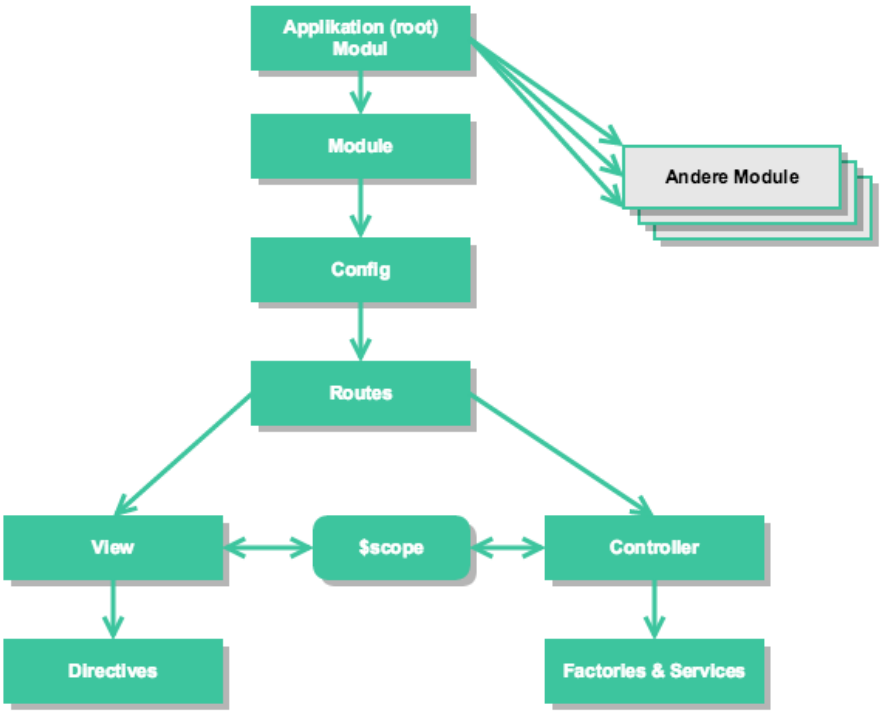


Abbildung 18 Architektur einer AngularJS-Applikation

3.3.2.1.1 Yeoman für Scaffolding

Innerhalb der Frontend-Applikation wurde Yeoman[15] verwendet. Yeoman integriert diverse Tools.

Abbildung 19 Verwendete Tools in Yeoman (Icons der offiziellen Yeoman-Webseite)

Für das Frontend relevant sind folgende Tools:

Tool	Beschreibung	Beispiel im Projekt
yo[16]	Scaffolding	Erstellung eines neuen AngularJS Controller `yo angular:controller qualityattribute`
Grunt[17]	Task Runner	Tests ausführen `grunt test`
Bower[18]	Package Manager	Neue Module installieren `bower install bootstrap`

[15] http://yeoman.io (zuletzt aufgerufen am 03.06.2015)
[16] https://github.com/yeoman/yo (zuletzt aufgerufen am 03.06.2015)
[17] http://gruntjs.com (zuletzt aufgerufen am 03.06.2015)

Karma[19]	Test Runner	Konfiguration des Test-Setups
Jasmine[20]	Test Specification	Implementation des Tests

Tabelle 13 Übersicht der Tools, welche im QUALI-T-Frontend verwendet werden

3.3.2.2 Routing

Beim Routing haben wir uns gegen das eingebaute ngRoute[21] entschieden. Unsere Entscheidung fiel auf ui-router.

Themengebiet	Frontend Modules	**Thema**	Technologie
Name	Auswahl Routing-Modul	**ID**	TEC-CLIENT-ROUT
Getroffene Entscheidung	ui-router		
Problemstellung	Welches Modul für Routing soll in der Frontend-Applikation gebraucht werden?		
Voraussetzung	Keine		
Motivation	Diese Entscheidung beeinflusst vor allem das Routingverhalten der Frontend-Applikation. Diese Entscheidung beeinflusst die gesamte Architektur der Applikation. Für die Entscheidung ist vor allem die Heterogenität mit bestehenden Applikationsarchitekturen (EEPPI) in den Vordergrund zu stellen. Ausserdem sollen etablierte Patterns (MVC mit Data-Binding) nicht manuell von den Entwicklern implementiert werden, sondern bereits existierende Komponenten verwendet werden können.		

Alternativen	• ngRoute
	  ○ Vorteil
	  ▪ Standardkomponente (in AngularJS eingebaut)		

Begründung	Als Routing-Modul wurde ui-router verwendet. Der Hauptgrund ist der Support von States. Das Modul ist neben ngRoute das meistverwendete Routing-Modul und wird von der Mehrheit der AngularJS-Module unterstützt.		
Annahmen	Keine		
Abgeleitete Anforderungen	Das Standardmodul ngRoute kann aus dem Projekt entfernt werden. Der Server bietet eine Service API (RESTful HTTP) an, sodass die AngularJS-Applikation mit Daten arbeiten kann (read and write).		
Verknüpfte Entscheidungen	Keine		

Designentscheidung 7 ui-router vs. ngRoute

Alle Routes werden gemäss ui-router als States erfasst und befinden sich in einer Datei (public/app/config/routes.js). Ein Auszug aus routes.js sieht folgendermassen aus:

[18] http://bower.io (zuletzt aufgerufen am 03.06.2015)
[19] http://karma-runner.github.io, zuletzt aufgerufen am 03.06.2015
[20] http://jasmine.github.io, zuletzt aufgerufen am 03.06.2015
[21] https://docs.angularjs.org/api/ngRoute, zuletzt aufgerufen am 11.06.2015

```
angular.module('qualitApp')
  .config(function ($stateProvider, $urlRouterProvider) {
    $urlRouterProvider.otherwise('/welcome');

    $stateProvider
      .state('welcome', {
        templateUrl: 'views/home/welcome.html',
        url: '/welcome',
        controller: 'HomeCtrl'
      })
      .state('login', {
        templateUrl: 'views/auth/login.html',
        url: '/login',
        controller: 'AuthCtrl'
      })
      // and some more
});
```

Code 3 Auszug aus der Routes-Konfigurationsdatei

3.3.2.3 MVC-Pattern

AngularJS beherrscht standardmässig das MVC-Pattern. In AngularJS sind die Komponenten folgendermassen gemappt:

MVC-Komponente	AngularJS Komponente
Model	JavaScript-Variablen in Controllern
View	HTML-Dateien
Controller	Controller (Aufruf der controller() Methode)

Tabelle 14 Mapping der MVC- und AngularJS-Komponenten

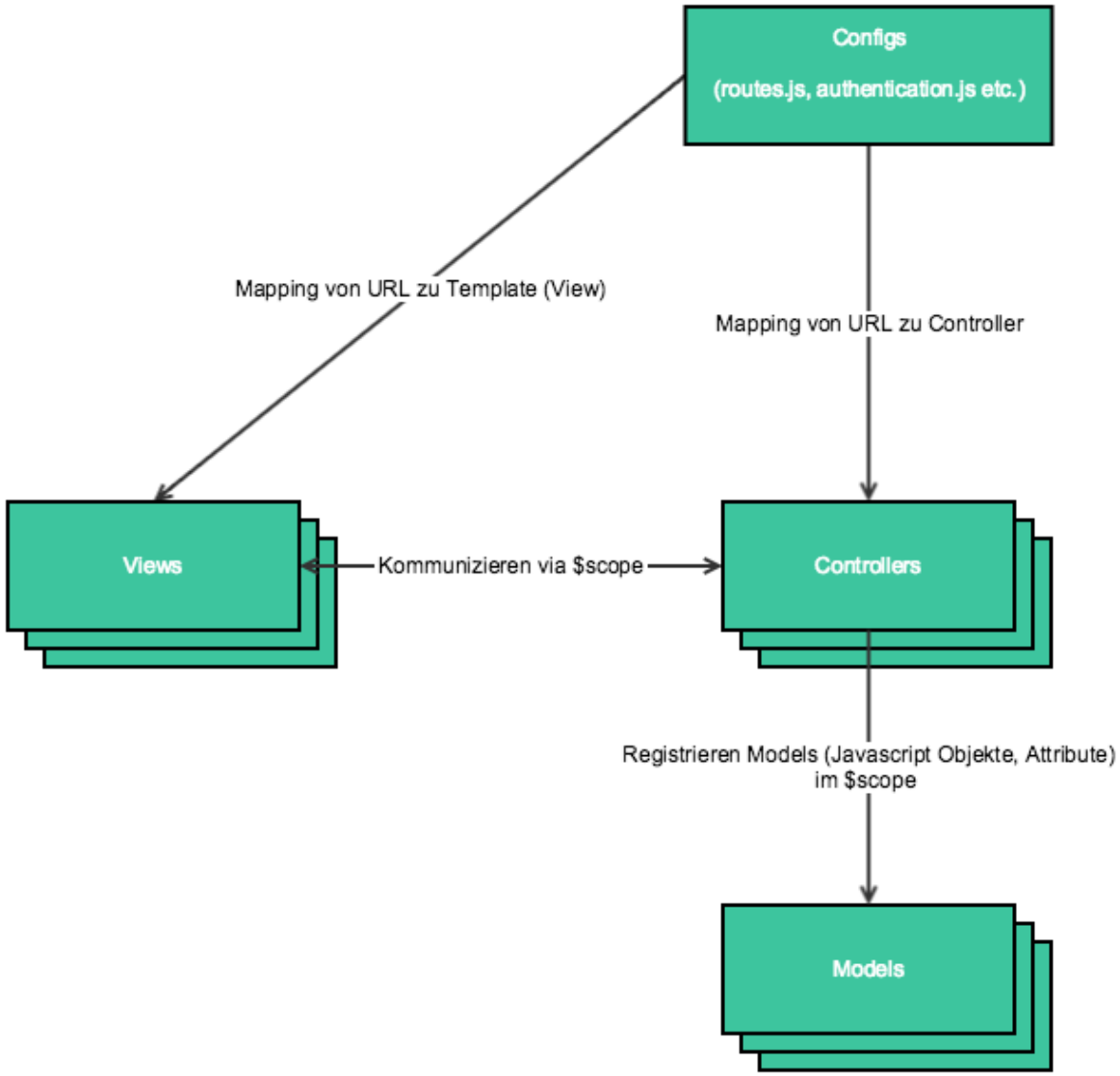


Abbildung 20 MVC Pattern in QUALI-T

3.3.2.3.1 Beispiel

Zur Veranschaulichung haben wir unten ein Beispiel aus der Frontend-Applikation visualisiert:

Szenario: Der eingeloggte Benutzer möchte auf die Seite mit der URL http://hostname:port/app/index.html#/authenticated/project/admin zugreifen. Beim Zugriff auf diese URL passiert Folgendes:

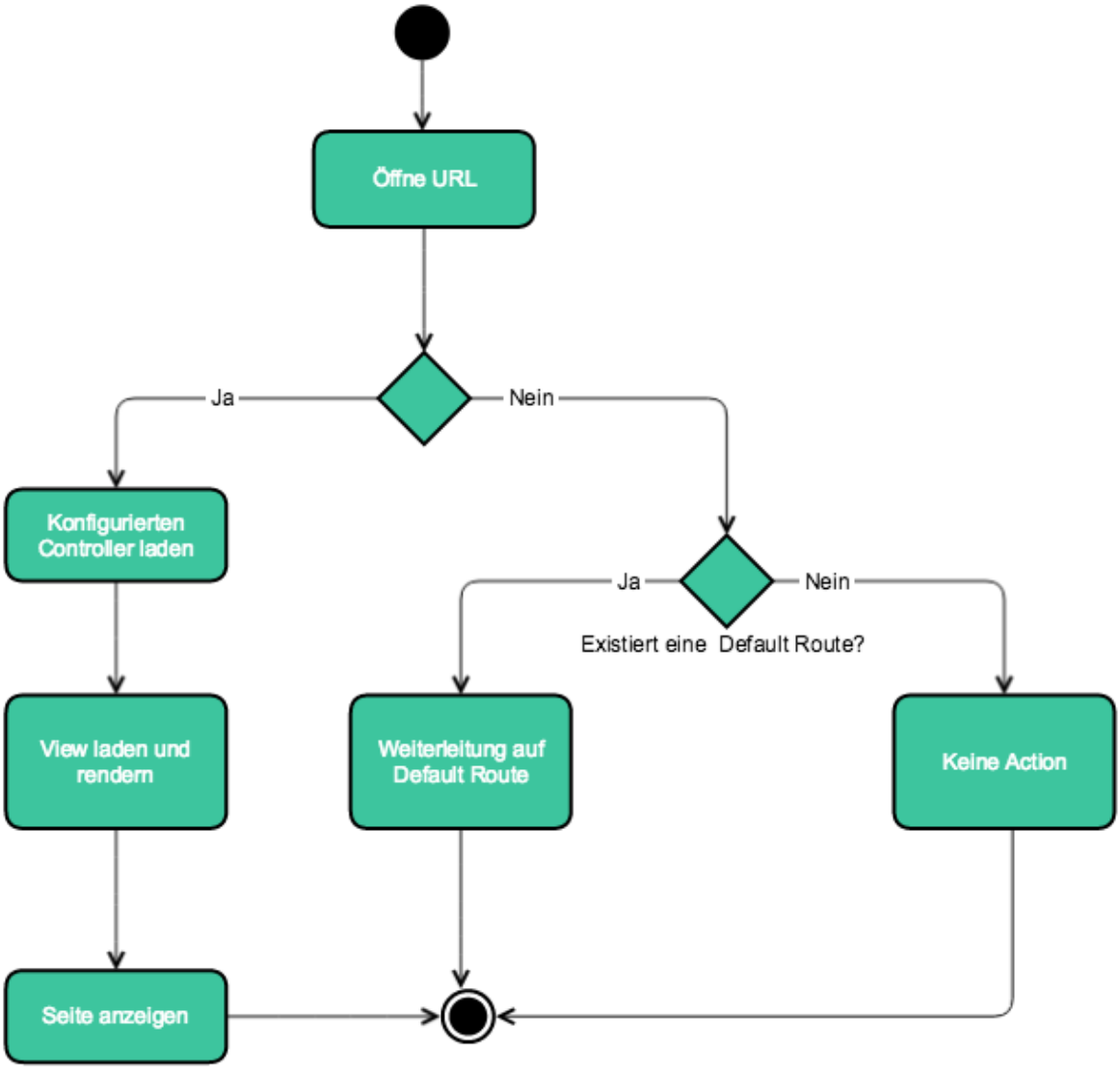


Abbildung 21 MVC-Workflow

3.3.2.4 Directives

Wiederverwendbare UI-Konstrukte können in Direktiven ausgelagert werden. Daraus ergeben sich dann neue HTML Tags. Dies haben wir für das Anzeigen eines Quality Attributes gebraucht.

3.3.2.4.1 Quality Attribute

Ein Quality Attribute wird an verschiedenen Stellen in der Applikation angezeigt:

- Auswahl der Quality Attributes für die Erstellung eines Catalogs / Projects
- Anzeige des Quality Attribute im Catalog / Project
- Preview von Quality Attributes

Durch die Direktive Quality Attribute kann das Anzeigen eines Quality Attributes mit folgendem HTML-Code erreicht werden:

```
<qa
  id="qa-{{catalogQa.id}}"
  categories="catalogQa.qa.categories"
  catalog-qa="catalogQa"
  qa="catalogQa.qa"
  context="editcatalog"
  variables="catalogQa.variables"
  editable="true"
  update-qa-function="loadCatalogQas">
</qa>
```

Code 4 HMTL-Code für das Einbinden der Quality-Attribute-Direktive

Diese Direktive ist vielseitig konfigurierbar. Die Parameter sind in der folgenden Tabelle beschrieben:

Parameter	Beschreibung
`id`	Standard-Attribut für DOM-Elemente.
`categories`	Kategorien des Quality Attribute. Werden als zusätzliche Information angezeigt.
`catalog-qa`	Da die Direktive zu einem Quality Attribute, aber auch zu einem CatalogQA gehören kann, wird dieses Attribut benötigt. Sie wird beim Editieren eines Catalog für die Anzeige des Zahnrads für das Editieren des Quality Attribute Template gebraucht.
`qa`	Das Quality Attribute Template mit der ID und der Beschreibung.
`context`	Das Verhalten der Direktive (z.B. Rendering von Buttons, Callback-Funktionen der Buttons) kann aufgrund des Kontexts variieren. Zum Beispiel hat ein Quality Attribute nur im „editproject“-Kontext einen Export-Button und einen Link für JIRA Issues. Mögliche Werte für context sind: • editcatalog • editproject • createproject • createcatalog • editqa
`variables`	Die Variablen, welche im Quality Attribute bzw. CatalogQA enthalten sind.
`editable`	Boolean-Wert, ob die Variablen veränderbar sind oder nicht.
`update-qa-function`	JavaScript-Callback-Funktion für die Kommunikation zwischen modalem Fenster und Main-Fenster. Wird aufgerufen, nachdem ein Quality Attribute editiert wurde.

Tabelle 15 Parameterliste der Quality-Attribute-Direktive

3.3.2.4.2 Filter (Categories)

Quality Attributes sollen nach ihren Categories gefiltert werden können. Da dieser Filter an verschiedenen Stellen (Create Catalog, Create Project) gebraucht wird, haben wir uns entschieden, diese UI-Komponente ebenfalls als Direktive auszulagern.

```
<filter>
  id="categories"
  callback =""
  hide-checkbox ="false"
  categories ="catList"
</filter>
```

Code 5 HMTL Code für das Einbinden der Filter-Direktive

Die Konfiguration für die Direktive sieht folgendermassen aus:

Parameter	Beschreibung
`id`	Standard-Attribut für DOM-Elemente.
`callback`	Callback-Funktion, welche aufgerufen wird, wenn die Checkbox angeklickt wird.
`hide-checkbox`	Boolean-Wert, ob eine Checkbox angezeigt werden soll oder nicht. Da die Direktive ebenfalls für die Administration der Categories selber gebraucht wird, wurde dieses Feld erstellt. In diesem Fall sind Checkboxes nicht erwünscht. Wenn die Direktive für die Filtrierung gebraucht werden soll, sollte eine Checkbox angezeigt werden.
`categories`	Categories als Baum mit Subcategories. Diese Categories werden iteriert und gerendert.

Tabelle 16 Parameterliste der Filter-Direktive

3.3.2.5 Services / Factories

Wiederverwendbare Logik wird in Services oder Factories abgebildet. Die für die Frontend-Applikation entwickelten Services werden hier detailliert vorgestellt.

3.3.2.5.1 apiService

Da die Frontend-Applikation die RESTful HTTP API der Backend-Applikation konsumiert, müssen diese Aufrufe implementiert werden. Um diese in verschiedenen Controllern wiederzuverwenden, haben wir einen Service entwickelt.

```
angular.module('qualitApp')
  .factory('apiService', function ($http, alerts, $rootScope) {
    var apiService = {};
    apiService.apiPath = "/api/";

    apiService.getQualityProperties = function () {
      return $http.get(this.apiPath + "qp")
        .success(function (data) {
          return data;
        })
        .error(function (data, status) {
          alerts.createError(status, "Quality Properties were not found.");
      });
    }

  return apiService;
});
```

Code 22 Beispiel einer Methode, welche die Backend API konsumiert

In Controllern wird der Service, wie alle Services in AngularJS, injiziert und kann danach gebraucht werden. Die Aufrufe erfolgen asynchron, um die Benutzeroberfläche nicht zu blockieren.

```
angular.module('qualitApp')
  .controller('QualitypropertyCtrl', function ($scope, apiService) {
    $scope.loadQualityProperties = function () {
      var loadPromise = apiService.getQualityProperties();
      loadPromise.then(function (payload) {
        $scope.qpList = payload.data;
      });
    }
});
```

Code 23 Injektion des apiServices im QualityPropertyController und Aufruf einer Service-Methode (asynchron)

3.3.2.5.2 alertService

Für Erfolgs- und Fehlermeldungen werden sogenannte Alerts (oftmals auch Toasts oder Pop-Up Notification [13] genannt) verwendet. Da diese Aufrufe immer gleich und wiederverwendbar sind, wurden sie in einen Service ausgelagert. Es gibt drei Typen von Alerts, welche angezeigt werden können:

Typ	Beschreibung
createSuccess(content)	Für Erfolgsmeldungen Der Parameter "content" wird dabei unterhalb vom Titel angezeigt.
createError(status, content)	Für Fehlermeldungen (von der Backend API) Der Parameter "status" wird in der HTTP Response vom Server zurückgesendet (z.B. 400 für Not Found). Der Parameter "content" wird ebenfalls in der HTTP Response vom Server zurückgesendet (z.B. die Fehlermeldung: Entity with ID XXX not found).
createLocalError(content)	Für Fehlermeldungen der QUALI-T-Applikation. Diese Fehlermeldungen werden unabhängig von der Backend API erstellt und angezeigt. Der Parameter "content" wird dabei unterhalb vom Titel angezeigt.

Tabelle 17 Typen von Alerts in alertService

Der Aufruf des Services (im Controller) ist einfach und erzeugt folgendes Resultat:

```
var alert = alerts.createSuccess('Project was successfully updated');
```

Code 24 Injektion des apiServices im QualityPropertyController und Aufruf einer Service-Methode (asynchron)

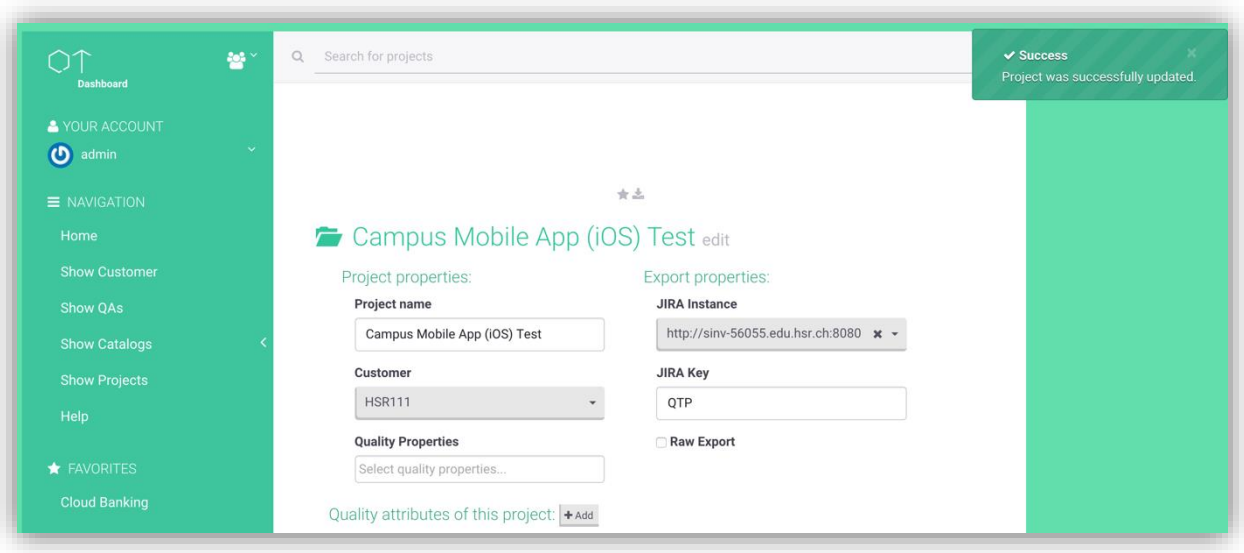


Abbildung 25 Beispiel eines Alerts (createSuccess)

3.3.2.5.3 qaTextService

Der Service qaTextService ist eine Hilfsklasse, um Beschreibungen von Quality Attributes zu verarbeiten. Im Service sind Methoden für Pattern Matching für das Extrahieren von Variablen aus dem Text enthalten. Sonstige Hilfsfunktionen, welche von mehreren Controllern verwendet werden können, sind im Service implementiert.

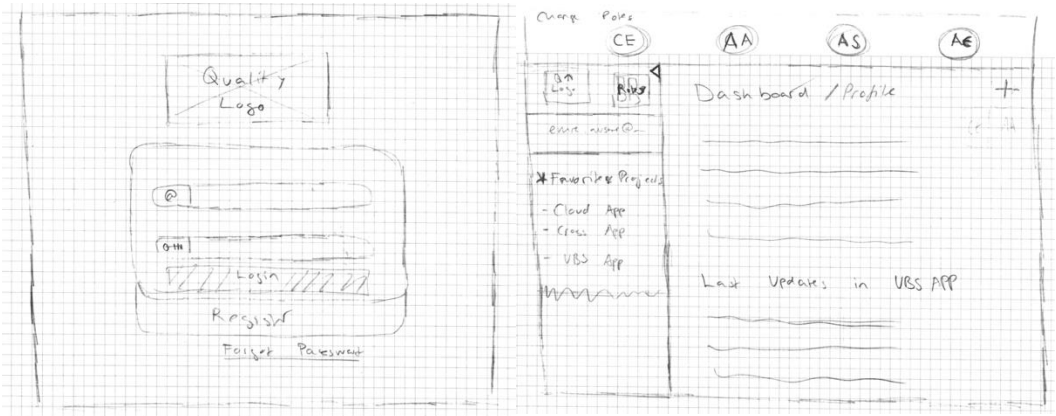
3.3.2.5.4 favoriteService

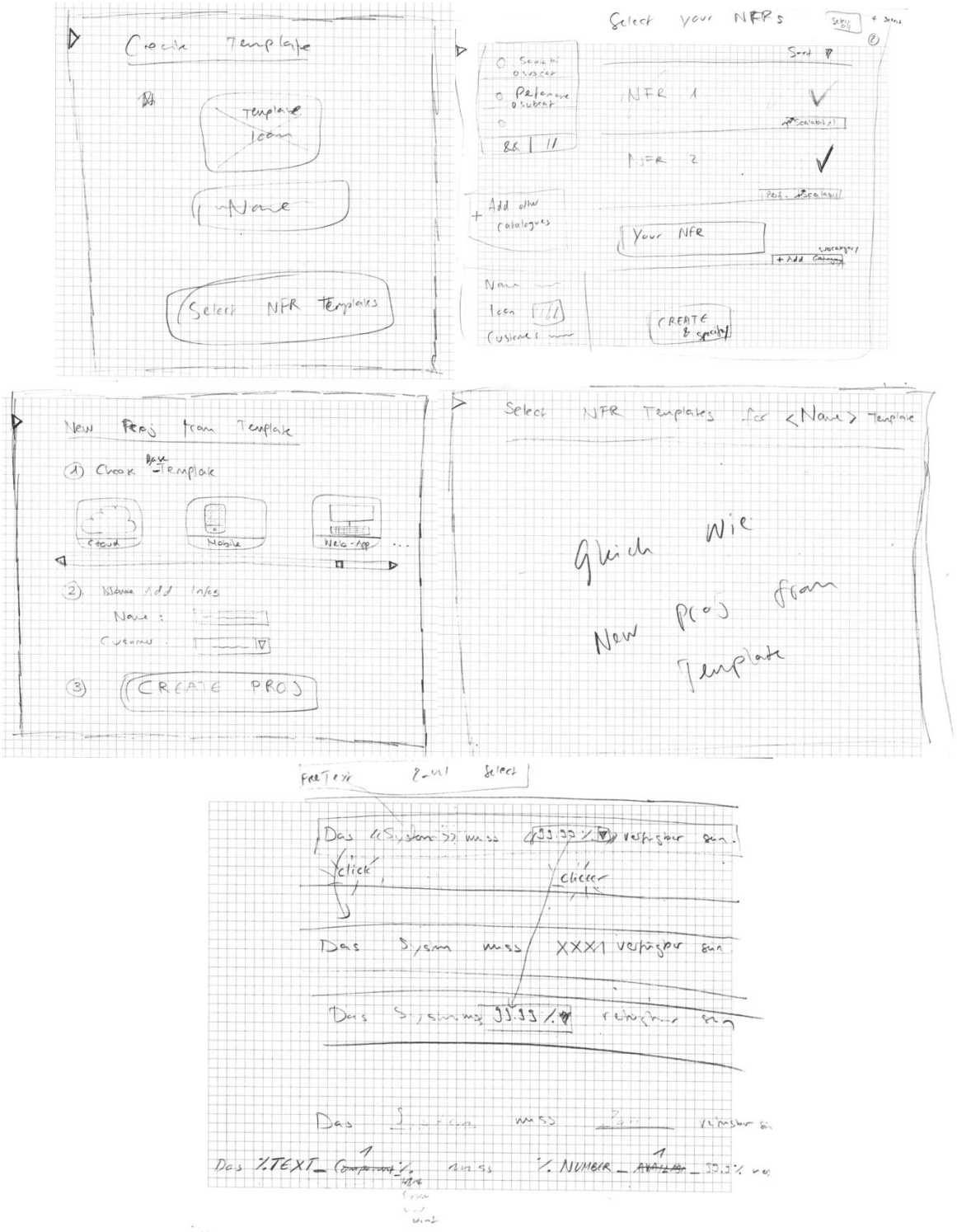
Der Benutzer kann Projekte zu seinen Favoriten hinzufügen. Im favoriteService wird die Funktionalität gekapselt, um herauszufinden, ob ein Projekt bereits favorisiert ist.

3.3.2.6 User Interface

Um eine passende Benutzeroberfläche zu designen haben wir im Vorfeld mit Paper Prototyping und Wireframes gearbeitet. Danach haben wir später daraus ein Bootstrap Theme erarbeitet. Den Fokus haben wir auf die Selektion von Quality Attributes und das Erstellen eines Katalogs gelegt.

3.3.2.6.1 Paper Prototyping





3.3.2.6.2 Wireframes

Wireframes haben wir digital in einem Bildbearbeitungstool erstellt. Diese Wireframes konnten wir dann mit der Marvel-App-Webapplikation zu einem interaktiven Prototyp[22] modellieren.

3.3.2.6.3 Login

[22]Link zum interaktiven Prototyp: https://marvelapp.com/6gjc46 (zuletzt aufgerufen am 03.06.2015)

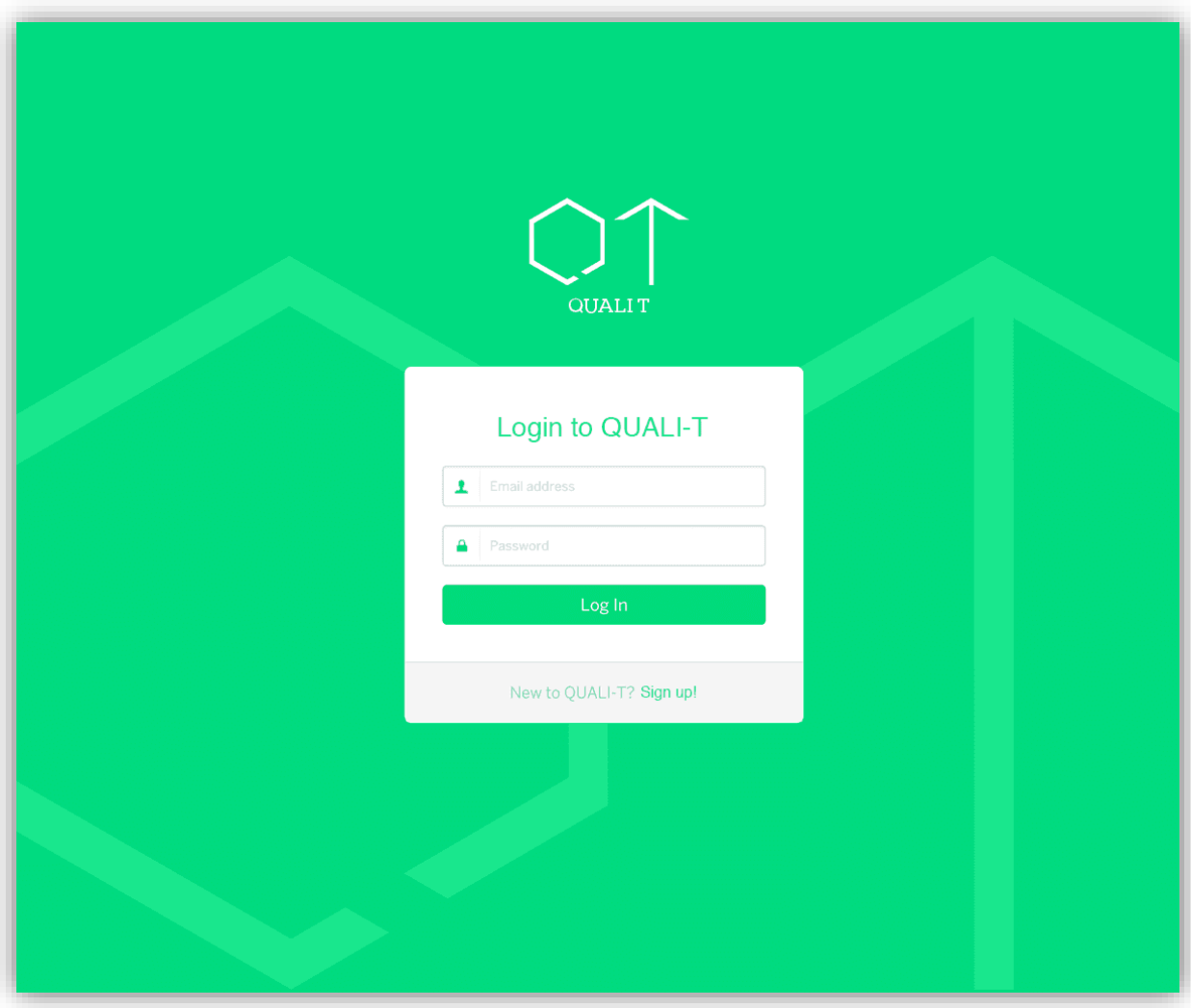


Abbildung 26 Wireframe des Login Screen

3.3.2.6.4 Dashboard

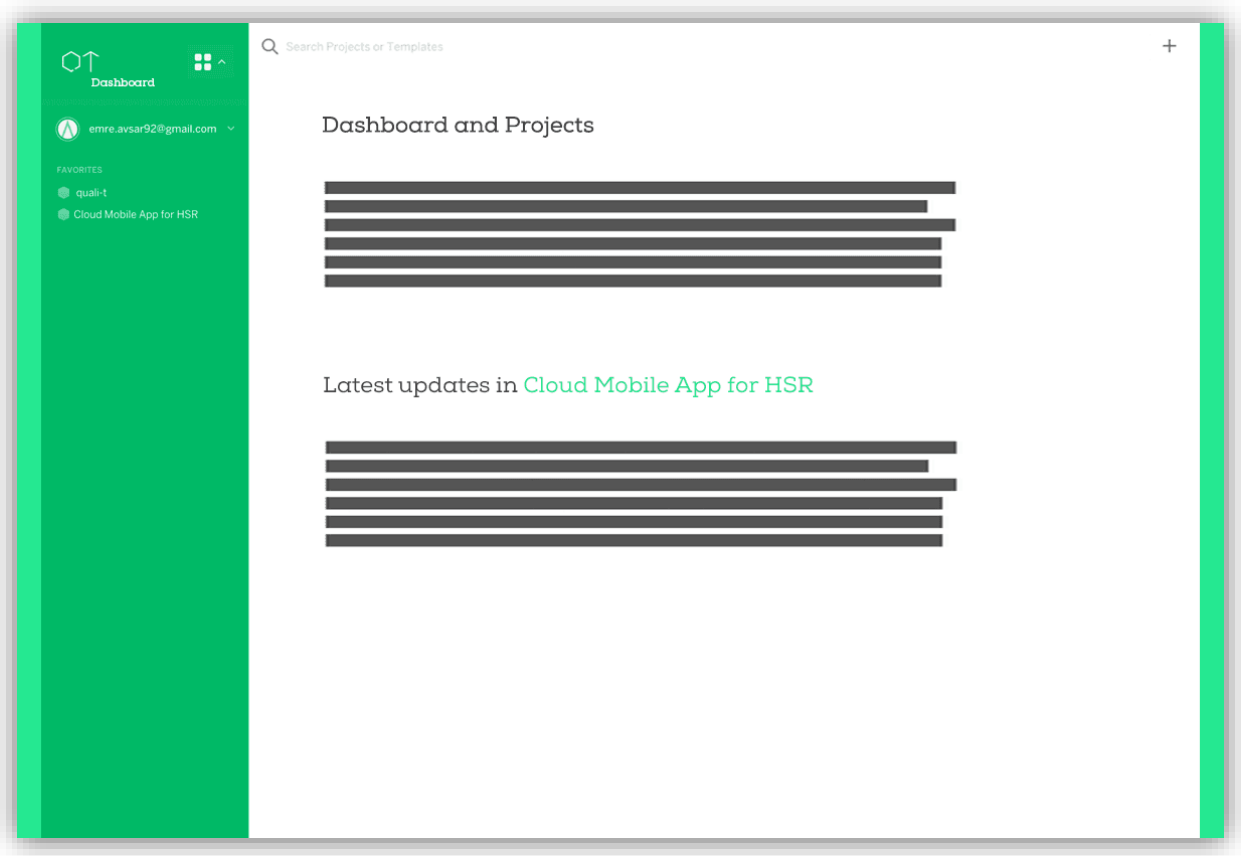


Abbildung 27 Wireframe des Dashboard Screen

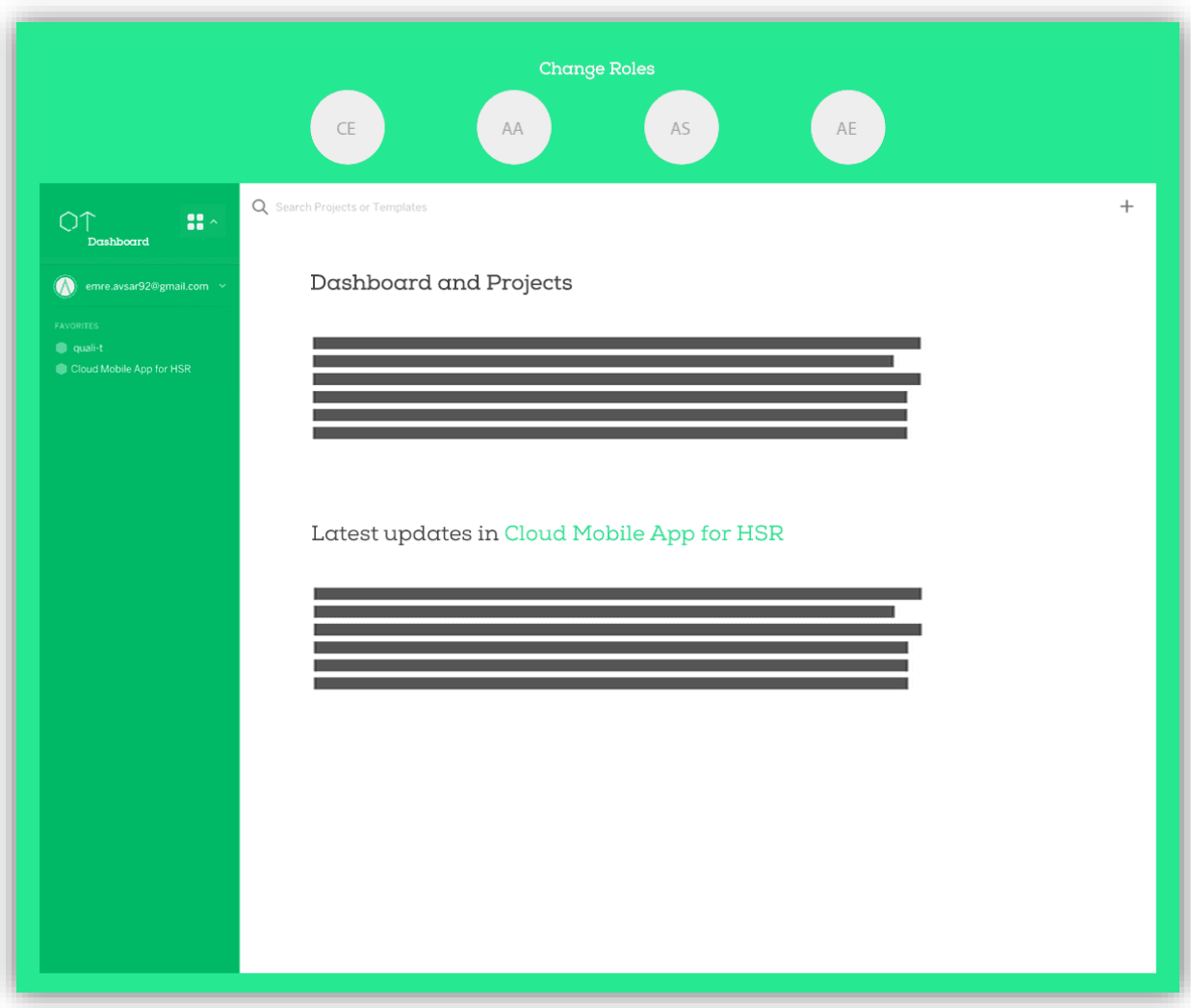


Abbildung 28 Wireframe des Dashboard Screen mit Role Changer (oben)

3.3.2.6.5 Catalog

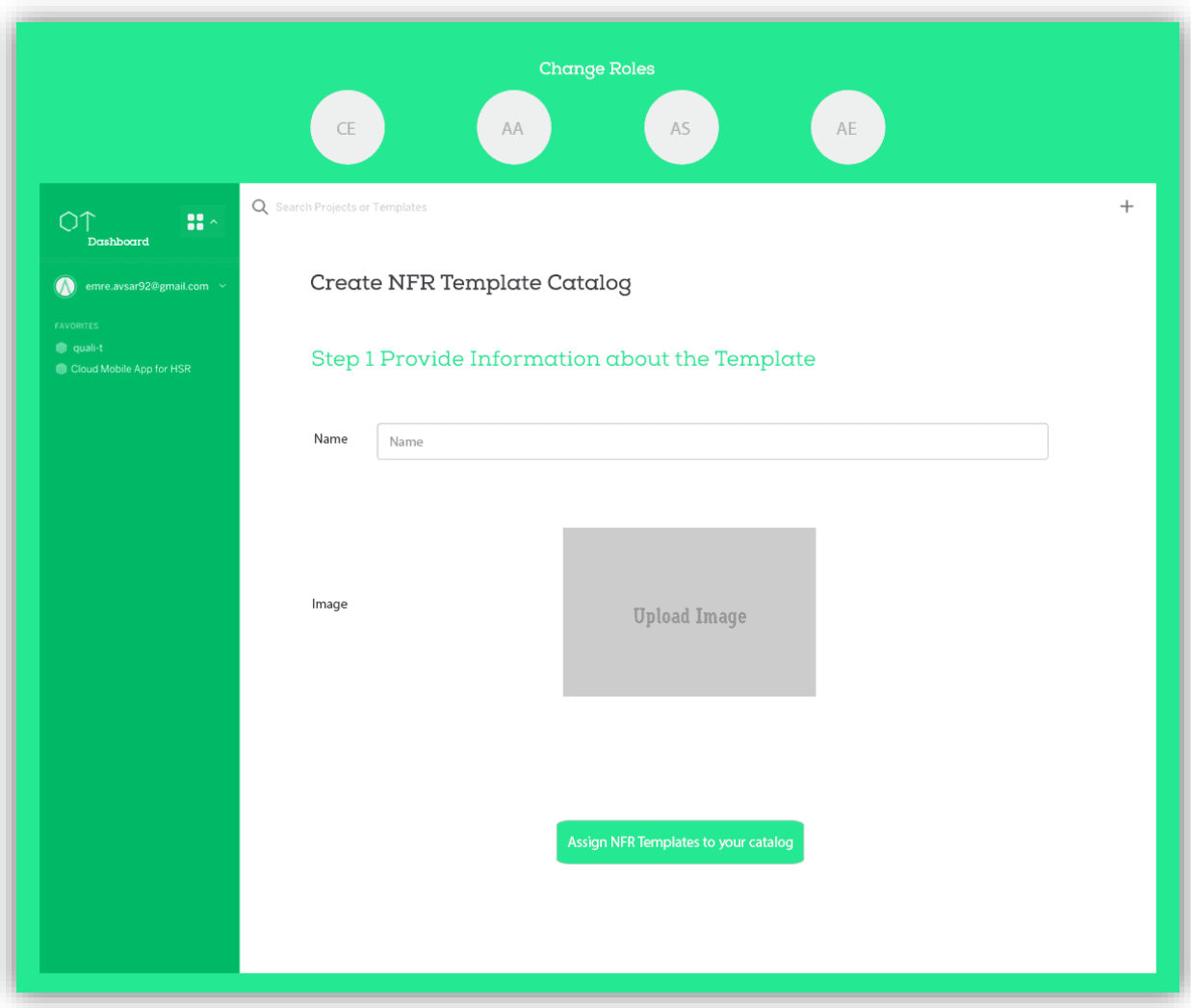


Abbildung 29 Wireframe des Create Catalog Screen, Schritt 1

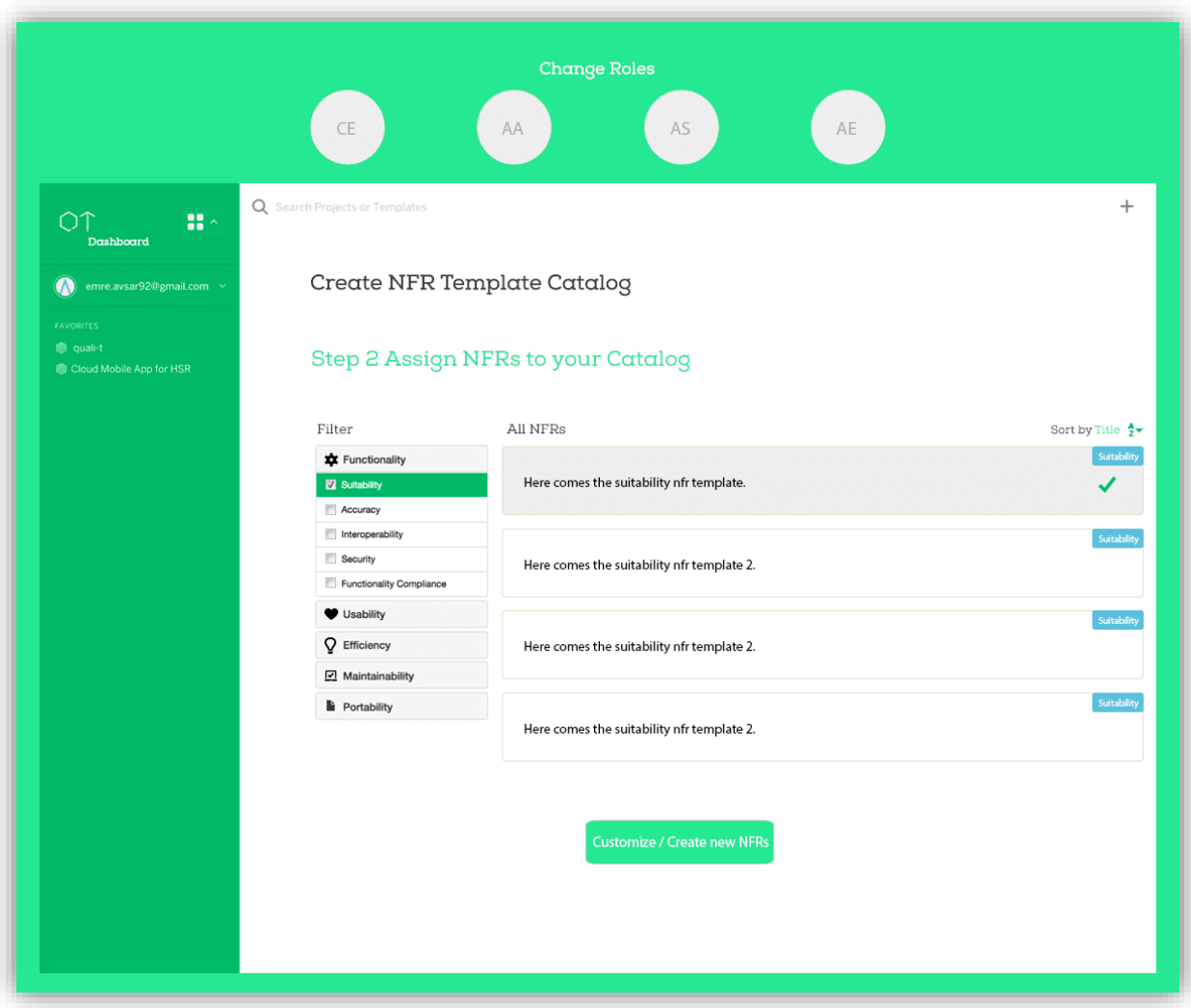


Abbildung 30 Wireframe des Create Catalog Screen, Schritt 2

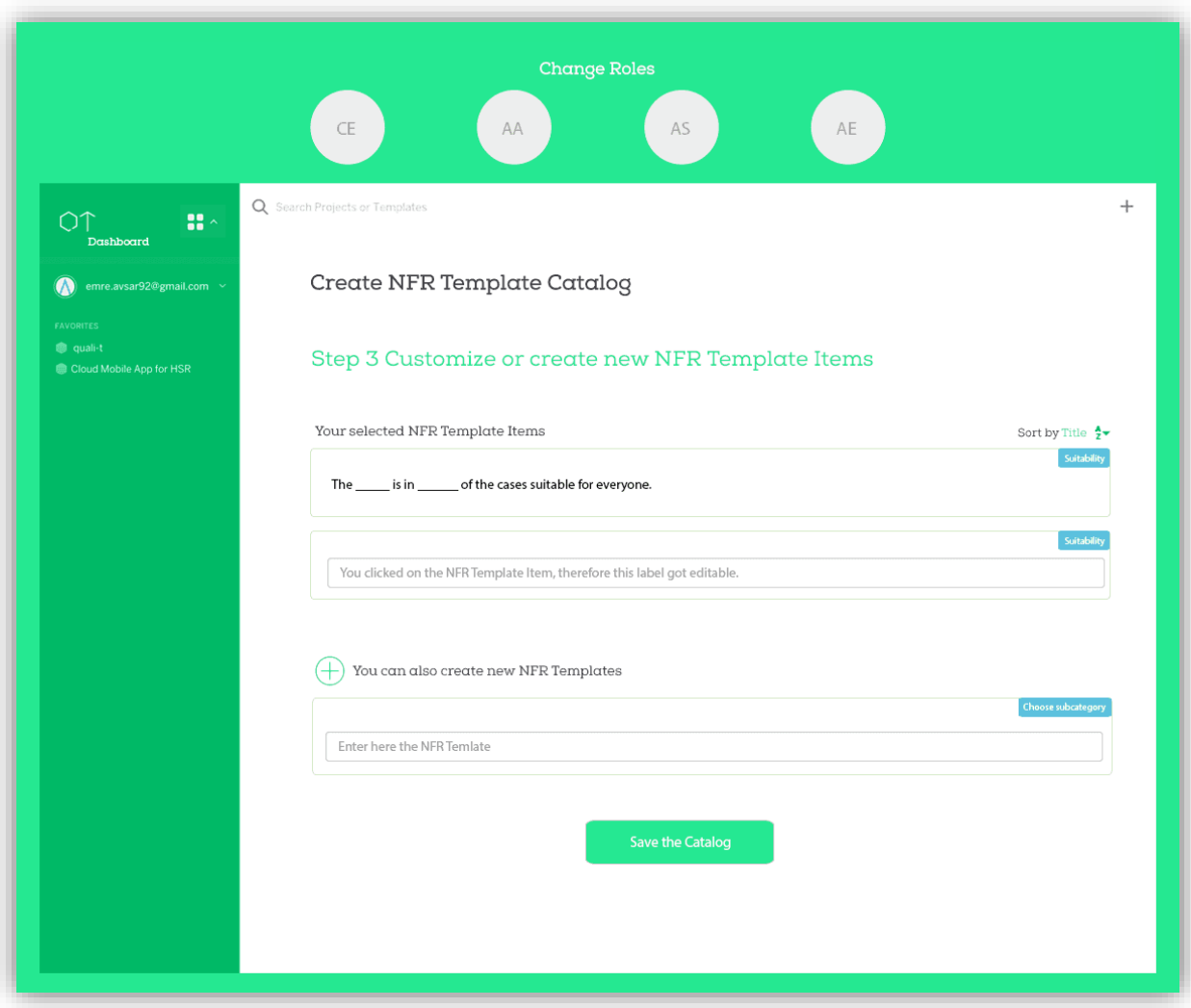


Abbildung 31 Wireframe des Create Catalog Screen, Schritt 3

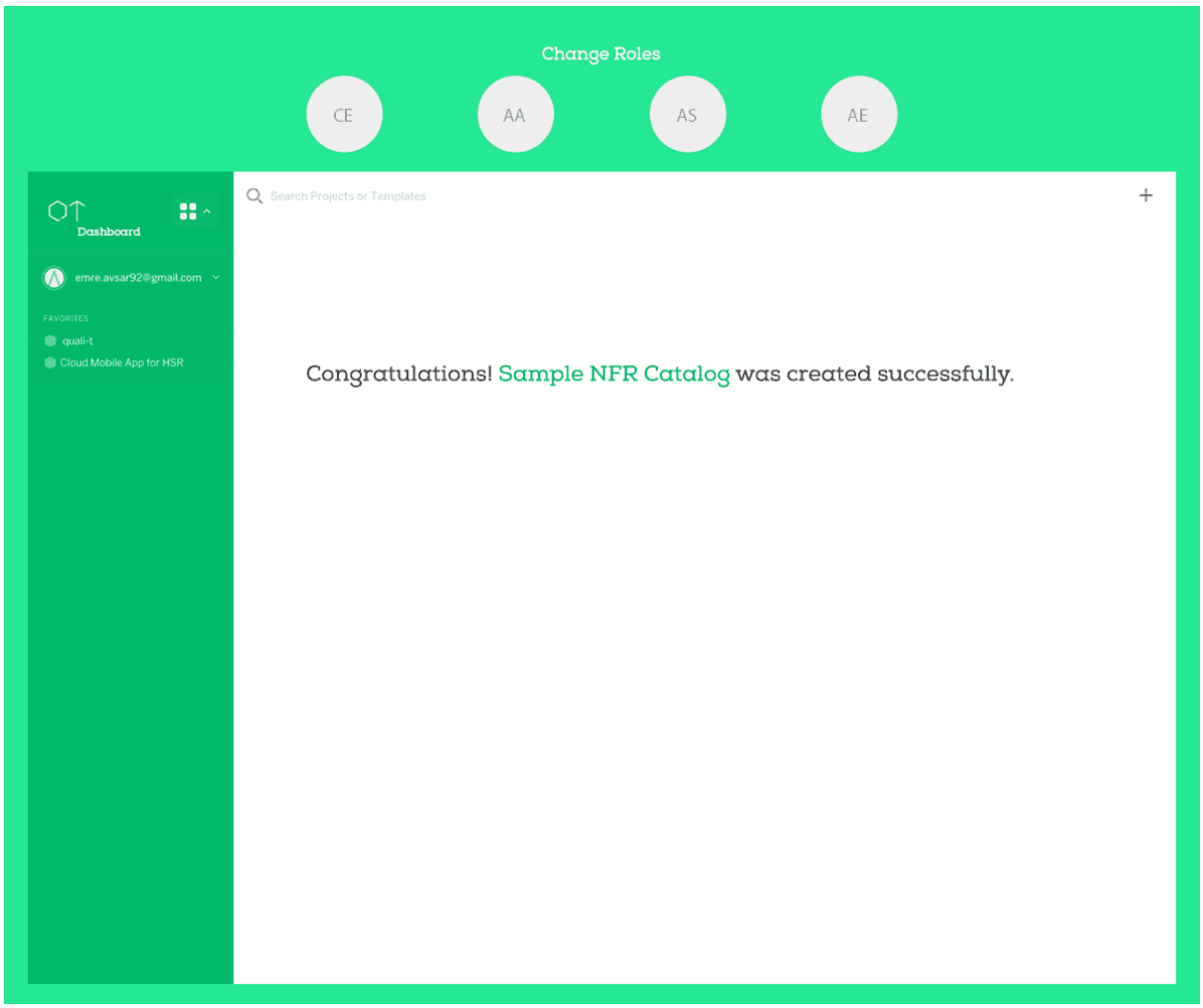


Abbildung 32 Wireframe des Create Catalog Screen, Schritt 4

3.3.2.6.6 Finales Design

Für das finale Design haben wir vor allem die Feedbacks der UI-Review-Tests und vorherige Reviews durch Herrn Zimmermann berücksichtigt. Um einen Vergleich mit den Wireframes machen zu können, ist unten ein Screenshot des Create Catalog Screen aus der Abgabeversion angefügt.

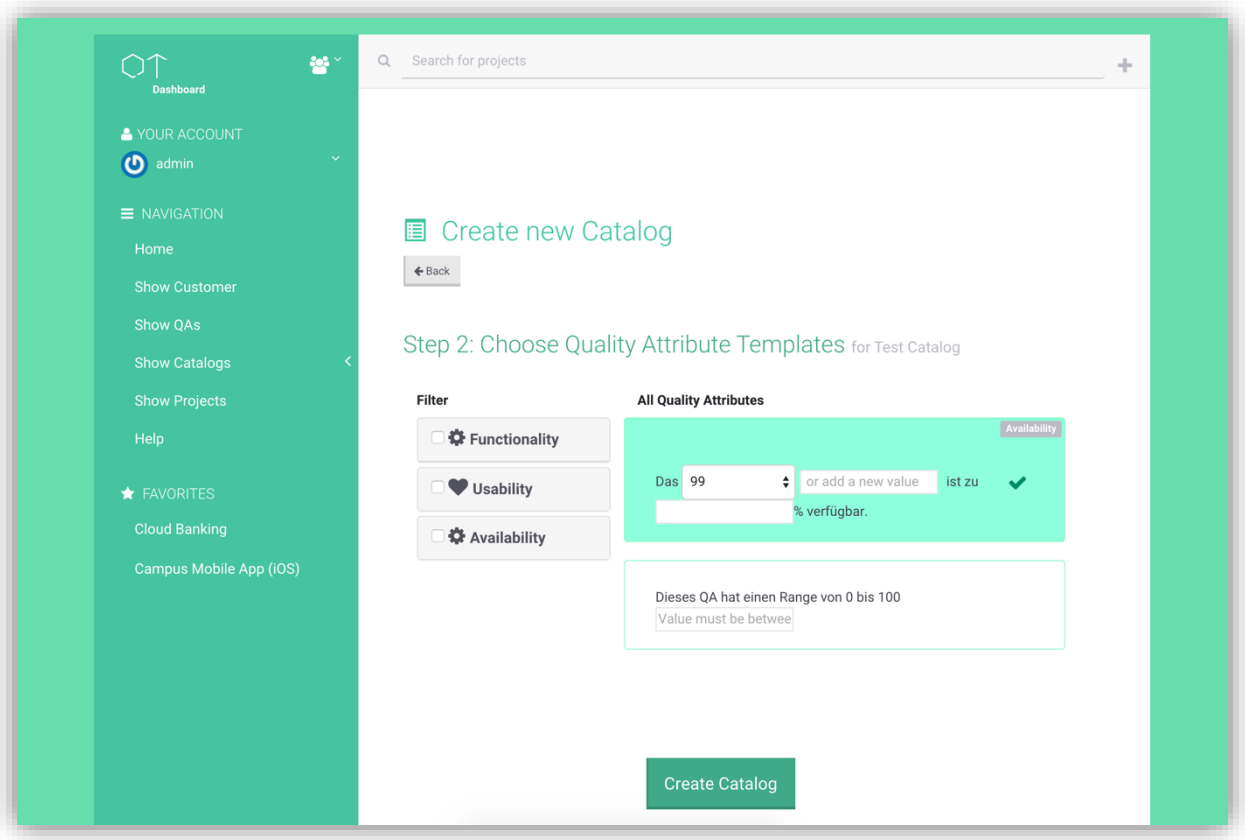


Abbildung 33 Screenshot des Create Catalog Screen aus der Abgabeversion von QUALI-T

3.3.3 Backend-Applikation

Die Backend-Applikation braucht das Play Framework (v2.3.8). Die Hauptaufgabe der Backend-Applikation ist die Bereitstellung einer RESTful HTTP/S API. Das Backend greift auf die Datenbank lesend und schreibend zu und bietet diese Daten als JSON an.

Entgegen des klassischen MVC-Patterns[23] wurden lediglich Model und Controller implementiert und die Views ausgelassen. Views werden nicht benötigt, da die Frontend-Applikation für das User Interface zuständig ist.

3.3.3.1 Architektur

Im Backend gibt es zusätzlich zu Models und Controllern noch weitere Komponenten. Die Verantwortlichkeiten der jeweiligen Packages sind im Kapitel 3.3.3.2 detailliert beschrieben.

[23] Weitere Informationen zum MVC-Pattern im Play Framework sind unter https://www.playframework.com/documentation/2.3.x/Anatomy#The-app/-directory verfügbar (zuletzt aufgerufen am 03.03.2015)

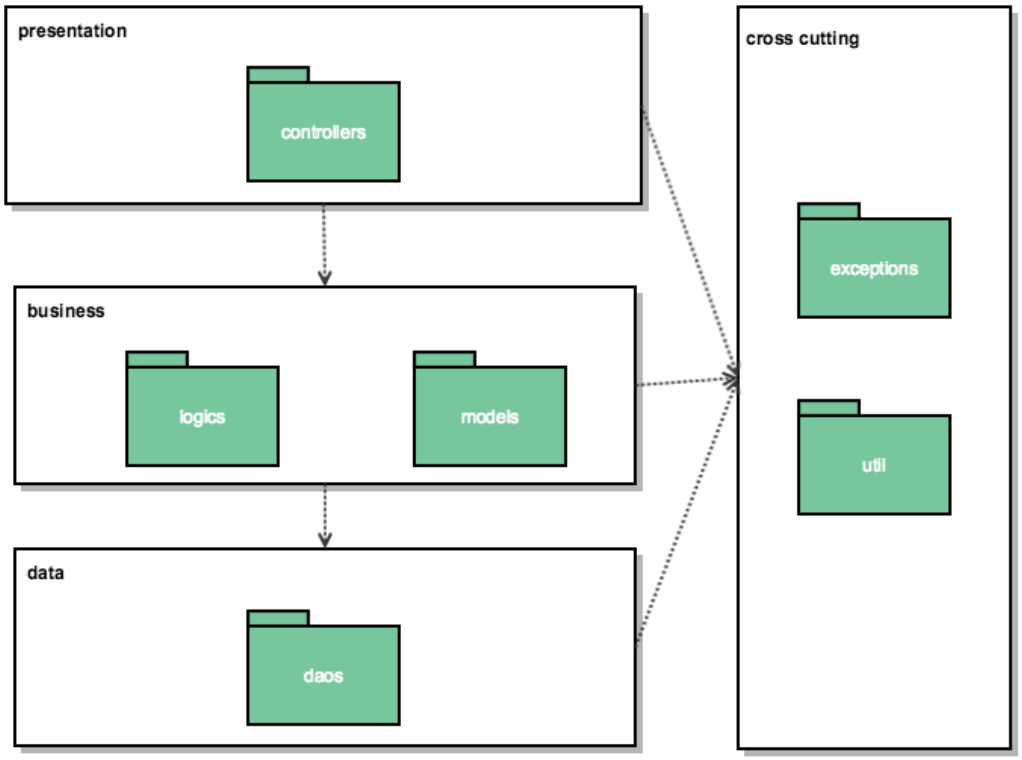


Abbildung 34 Logisches Modell in der Backend-Applikation

3.3.3.2 Klassenmodell

In diesem Kapitel werden die Packages mit den dazugehörigen Klassen im Backend vorgestellt.

3.3.3.2.1 Controllers Package

In diesem Package sind alle Controller-Klassen zusammengefasst. Alle Controller-Klassen erben vom play.mvc.Controller und gehören bereits zum Application Logic Layer. Sie sind dafür zuständig, die RESTful HTTP Requests mit JSON Body entgegenzunehmen und an die richtigen Logic-Klassen weiterzuleiten. Die Controller-Klassen nehmen auch die Antworten der Logic entgegen, wandeln sie in ein JSON um und schicken sie als HTTP Responses an den Aufrufer zurück. Sie fangen auch mit Hilfe des implementierten *ExceptionHandlingInterfaces* alle Exceptions der Logic- und DAO-Klassen ab. Das Exception Handling ist im Kapitel 3.1.10.3 detailliert dokumentiert. Die *Search*-Klasse implementiert dieses nicht, da bei der Suche keine Exceptions auftreten können. Das gleiche gilt für die *Application-Klasse.*

Die Controller-Klassen sind jeweils für die Modifikation von einer oder zwei Model-Klassen zuständig. Die Controller-Klassen wurden teilweise zusammengefasst, zum Beispiel Project und QA Instances, da QA Instances direkt von einem Project abhängen. Die Klassen und Fields können dem nachfolgenden Klassendiagramm entnommen werden. Aus Platzgründen wurde hier auf die Auflistung der Methoden verzichtet.

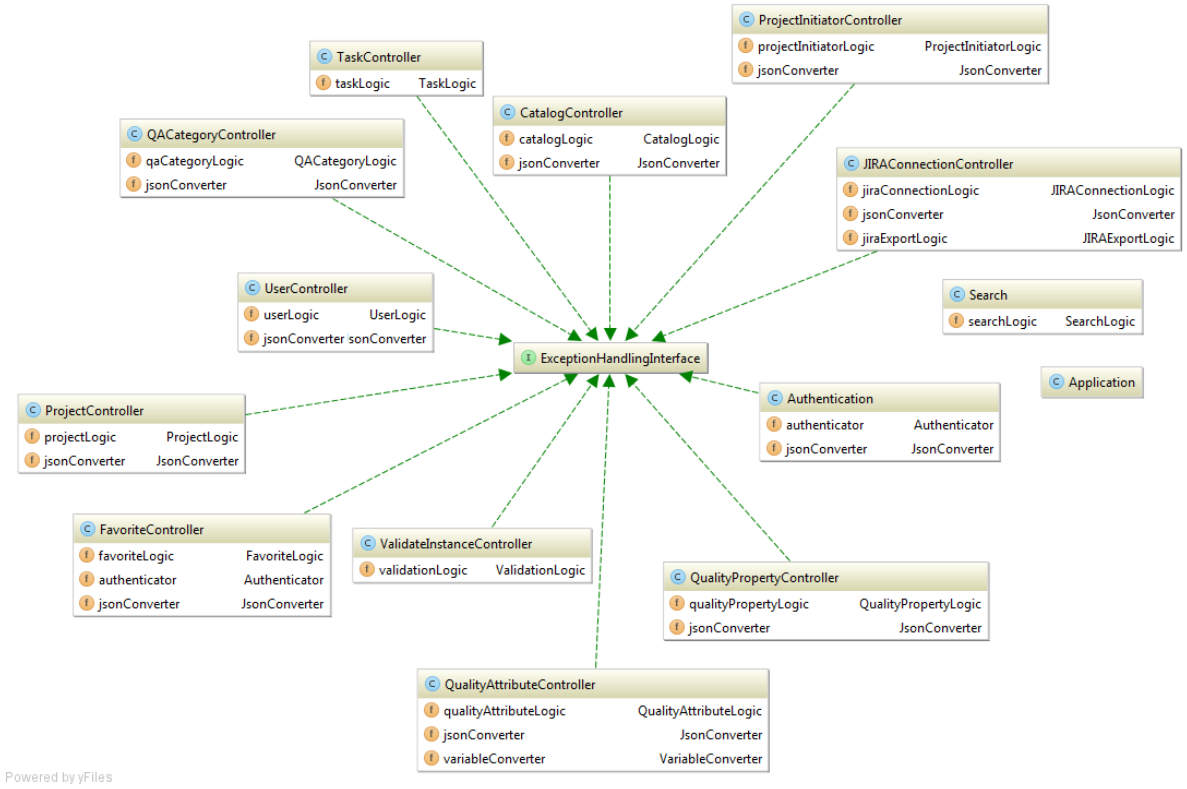


Abbildung 35 Klassendiagramm Controller Package ohne Methoden

3.3.3.2.2 Logics Package

In diesem Package werden alle Logic-Klassen zusammengefasst. Diese Klassen stellen die effektive Business Logic dar. Sie greifen über mit Dependency Injection injizierte DAO-Klassen auf die DB zu und führen alle CRUD- und Export-Funktionen aus. Pro Controller-Klasse existiert grundsätzlich eine Logic-Klasse. Es gibt aber auch einige zusätzliche, welche die Authentifizierungs-, die Export- oder die Fuzziness- und Statistik-Validation-Logic enthalten. Die Logics werfen auch alle Exceptions, die nicht mit dem DB-Zugriff zusammenhängen. Dazu gehören beispielsweise Exceptions, die bei fehlerhafter Passwort-Eingabe oder bei fehlenden Attributen des HTTP Requests auftreten können.

Das nächste Bild zeigt eine Übersicht aller Logics-Klassen und ihren Fields. Aus Platzgründen und zur Bessern Verständlichkeit wurde auch hier auf die Auflistung der Methoden verzichtet.

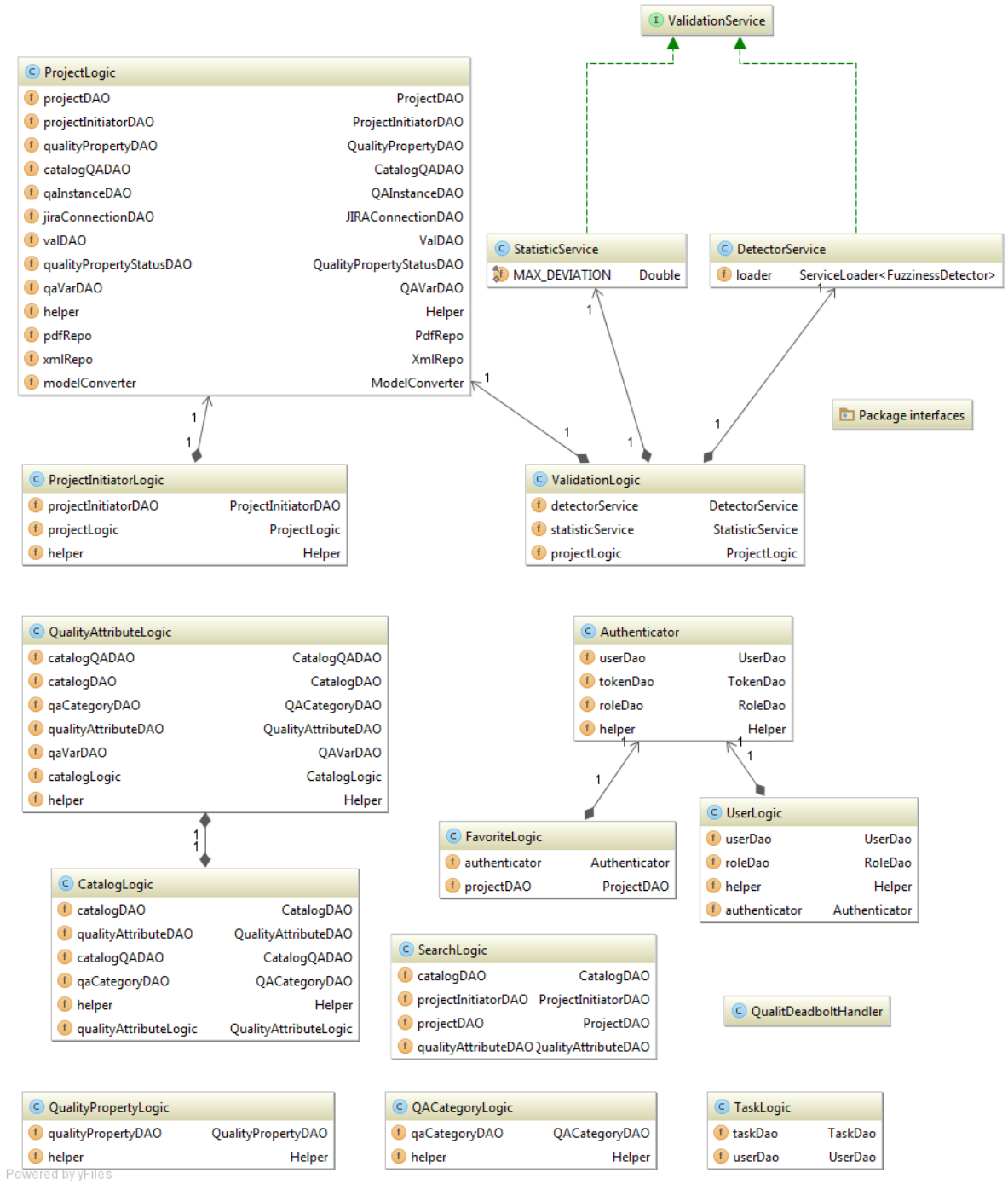


Abbildung 36 Logics-Package-Klassendiagramm

3.3.3.2.2.1 Interfaces Package

Ein Subpackage von Logics ist das Interfaces Package. In diesem Package sind alle Klassen, welche die Logik für den Export in JIRA, PDF oder XML enthalten. Darin sind auch die beiden Klassen ProjectInitiator und Project, welche ähnlich den entsprechenden Model-Klassen sind, jedoch nur die für den PDF-/XML-Export notwendigen Variablen, welche mit javax.xml.bind.annotation-Annotationen versehen sind, enthalten. Diese Annotationen sind für die Generierung des XML-Files notwendig.

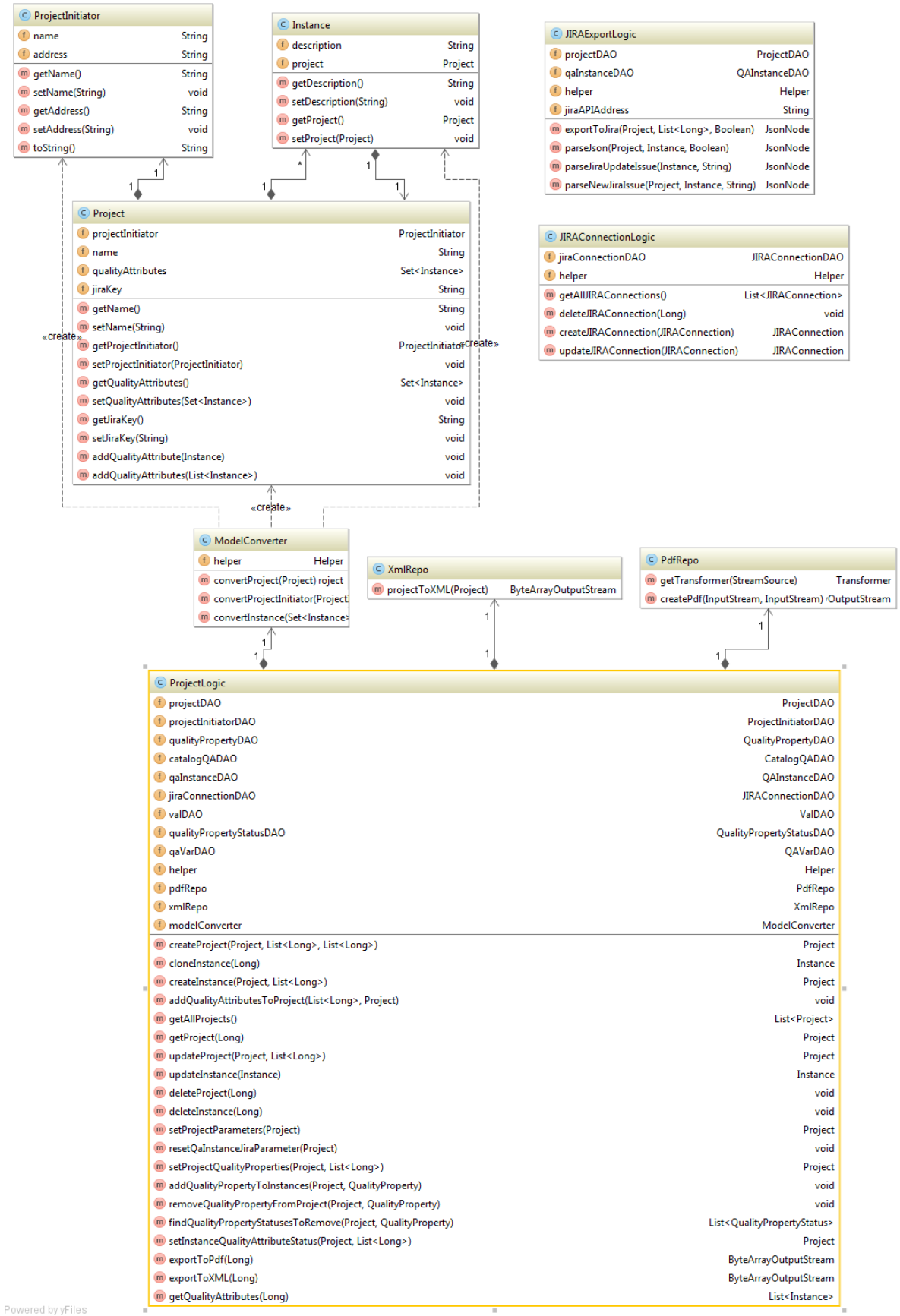


Abbildung 37 Interfaces-Package-Klassendiagramm

3.3.3.2.3 Models Package

In diesem Package sind alle Models zusammengefasst. Die Model-Klassen beinhalten keine Logic, sondern sind für die Abbildung der Objekte auf die Datenbank zuständig. Es existiert eine *AbstractEntity*-Klasse, welche die Generierung der Id-Variabel übernimmt. Die Id ist notwendig für das Persistieren der *Models*. Alle *Models* erben darum von der *AbstractEntity*.

Mit javax.persistence-Annotationen wird in diesen Klassen gesteuert, was mittels Hibernate auf die Datenbank abgebildet wird. Daneben werden auch com.fasterxml.jackson.annotation-Annotationen verwendet. Diese steuern, welche Variablen bei der Serialization zu JSON verwendet oder ignoriert und welche Verbindungen zwischen den Models aufgelöst werden.
Um Hibernate und JSON Serialization erfolgreich zu nutzen, wird in den Models für jede Variable GETTER und SETTER benötigt.

Aus Platzgründen wurde hier auf die Auflistung der Methoden sowie der Fields verzichtet.

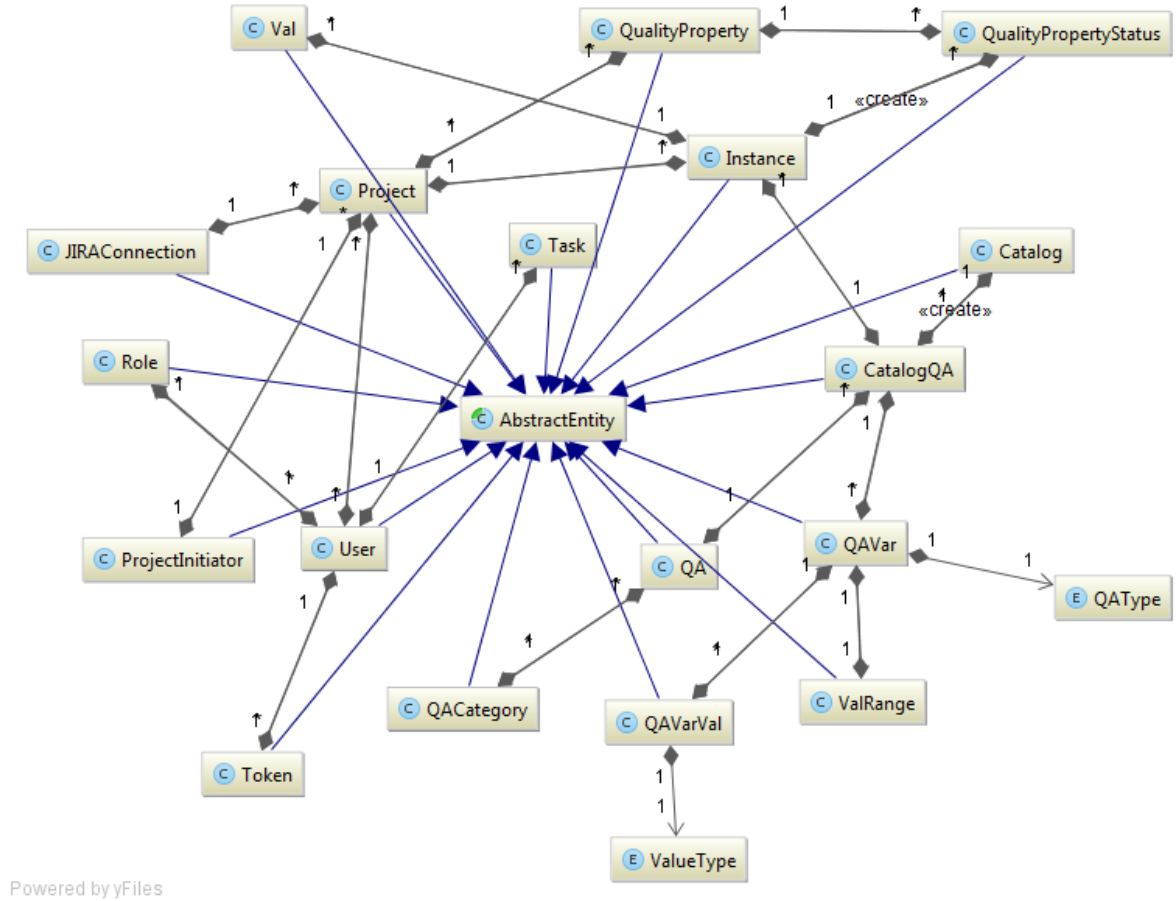


Abbildung 38 Models-Package-Klassendiagramm

3.3.3.2.4 DAO Package

Im DAO Package sind alle DAO-Klassen. Diese Klassen erben alle von der *AbstractDAO*-Klasse und sind für den Zugriff auf den Persistence Layer zuständig. Das heisst, dass alle Structured Query Language (SQL) Queries über die DAO-Klassen spezifiziert und mit Hilfe des JPA EntityManager ausgeführt werden. Die DAOs können Exceptions werfen, die beim Zugriff auf die Datenbank vorkommen. Dazu gehört zum Beispiel die *EntityNotFoundException*.

Pro Model, für welches ein direkter Datenbankzugriff benötigt wird, existiert eine DAO-Klasse.

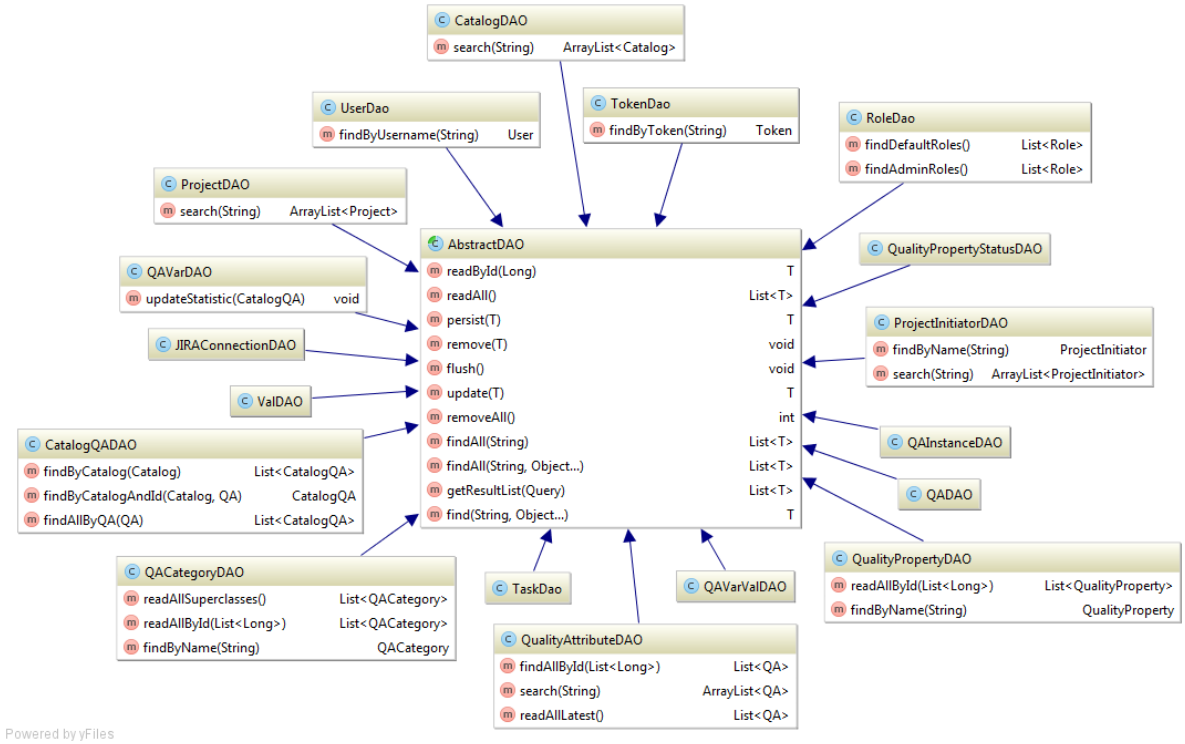


Abbildung 39 DAO-Package-Klassendiagramm

3.3.3.2.5 Hibernate Package

In diesem Package befindet sich die Klasse ImprovedPostgresSQL82Dialtect. Diese wird von Hibernate benötigt, um die Datenbankinhalte bei Bedarf sauber zu löschen, damit anschliessend das SQL Import Script mit den initialen Daten geladen werden kann.

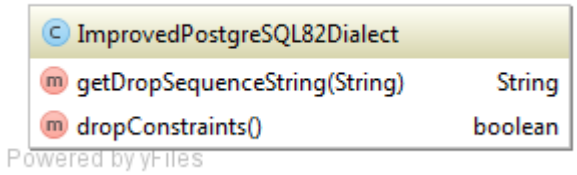


Abbildung 40 Hibernate-Package-Klassendiagramm

3.3.3.2.6 Util Package

Das Util Package beinhaltet Converters für die Controllers und eine Hilfskasse für die Logics. Die QUALI-T API nimmt diverse HTTP Requests mit JSONs entgegen. Die VariableConverter-Klasse ist dafür zuständig, alle CatalogQA- und QA-Variabel-Informationen aus einem JSON auszulesen und in die dafür geeigneten Klassen aus dem Model Package zu überführen. Die JSONConverter-Klasse ist dafür zuständig, alle anderen JSON-Inhalte auszulesen und in geeignete Klassen zu überführen. Dies sind Klassen aus dem Model Package oder Listen. Die *Helper*-Klasse beinhaltet einige Hilfsfunktionen, die in den Logics oder in den Converter-Klassen benötigt werden. Dazu gehören zum Beispiel eine String-Validierungsmethode und eine Methode, die Listen mit String-Objekten in eine Liste mit Long-Objekten konvertiert. In der *GlobalVariables*-Klasse sind ausserdem Konstanten definiert, die globale Gültigkeit haben und im ganzen Projekt wiederverwendet werden.

Das folgende Bild zeigt die Klassen mit den dazugehörigen Methoden:

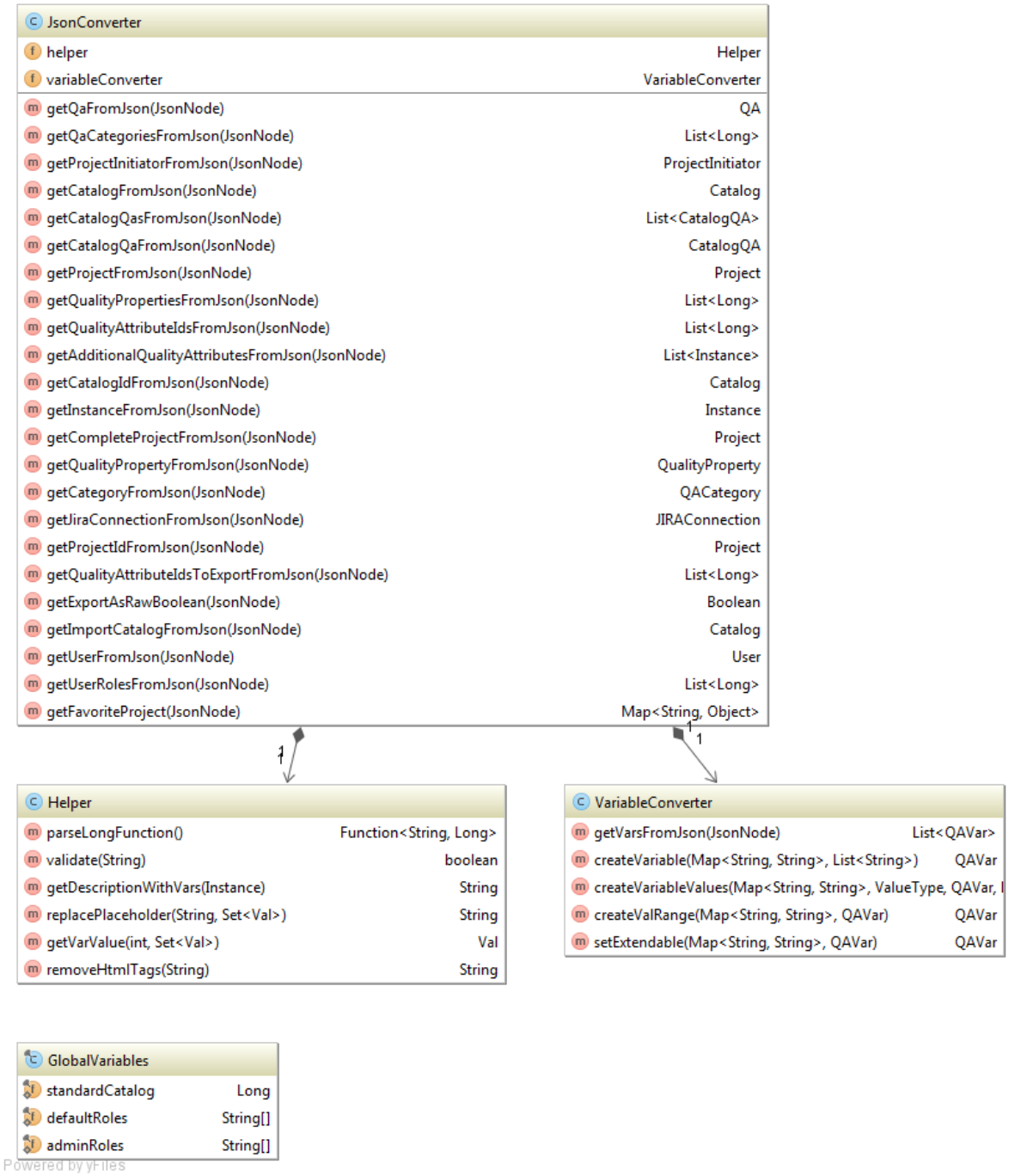


Abbildung 41 Util-Package-Klassendiagramm

3.3.3.2.7 Exceptions Package

Hier werden alle unsere selbstgeschriebenen Exceptions zusammengefasst. Alle erben von der *AbstractException*-Klasse, welche wiederum von der offiziellen *Exception*-Klasse erbt. Die Exceptions beinhalten zusätzlich den HTTP Status Code, welcher bei einer Exception von den Controllern an den Aufrufer zurückgegeben wird. Nachfolgend sind alle Exception-Klassen aufgeführt. Weitere Informationen sind im Kapitel 3.1.10 zu finden.

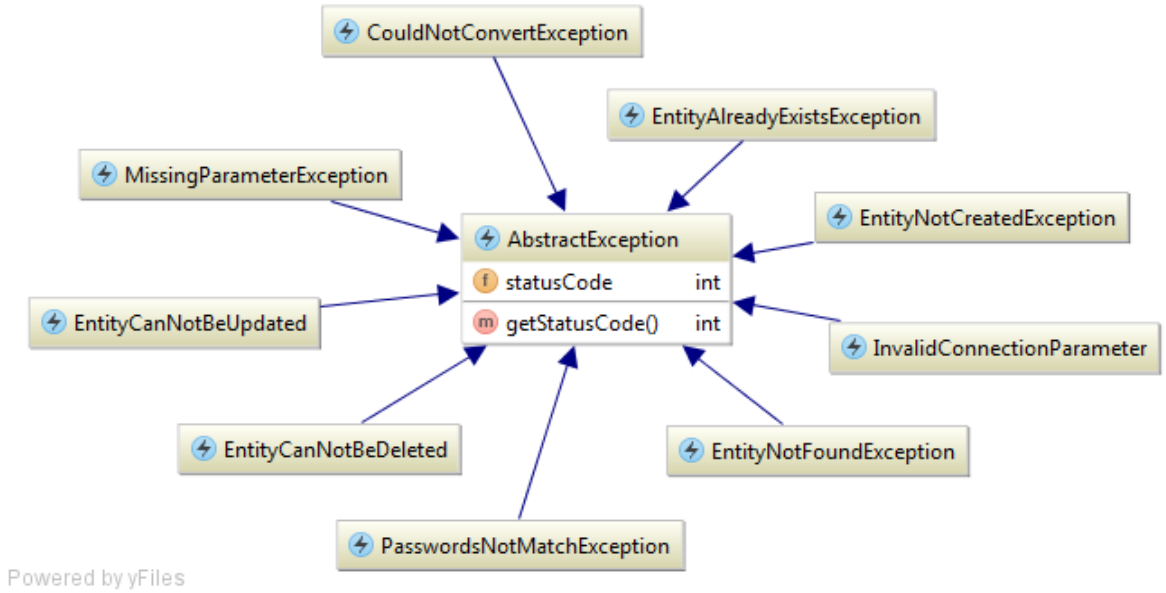


Abbildung 42 Exception-Package-Klassendiagramm

3.3.3.3 Datenbank-Schema

Das Datenbank-Schema, inklusive aller Constraints und Entity-Assoziationen, wird von Hibernate direkt aus den entsprechenden Annotationen in den Model-Klassen generiert. QUALI-T verwendet dafür die javax.persistence-Annotationen[24] gemäss der folgenden Tabelle:

Annotation	Verwendung
@Entity	Spezifiziert die Klasse als Entity
@Table	Spezifiziert die primäre Tabelle für die annotierte Entity (muss in Zusammenhang mit @Entity verwendet werden)
@Inheritance	Spezifiziert die Vererbungsstrategie für eine Entity-Klassenhierarchie (wird für die AbstractEntity benötigt)
@Id	Spezifiziert den PrimaryKey einer Entity
@SequenceGenerator	Definiert einen PrimaryKey Generator, welcher von @GeneratedValue verwendet werden kann
@GeneratedValue	Spezifiziert die Generierungsstrategie eines PrimaryKey
@OneToOne	Spezifiziert eine 1:1-Assoziation zwischen zwei Entities
@OneToMany	Spezifiziert eine 1:*-Assoziation zwischen zwei Entities
@ManyToOne	Spezifiziert eine *:1-Assoziation zwischen zwei Entities (wird auf der Gegenseite der @OneToMany-Referenz verwendet)
@ManyToMany	Spezifiziert eine *:*-Assoziation zwischen zwei Entities
@Enumerated	Definiert das Mapping für einen Enumerated-Typ (wird bei der Variablen gesetzt, welche auf ein Enum referenziert)
@JoinTable	Spezifiziert das Mapping zwischen zwei Tabellen (wird bei einigen ManyToMany-Assoziationen verwendet)
@JoinColum	Wird in @JoinTable verwendet, um die ForeignKeys der zu referenzierenden Assoziationen zu verwenden

Tabelle 18 Verwendete JPA-Annotationen

[24] Dokumentation der javax.persistence-Annotationen: http://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html (zuletzt aufgerufen am 10.06.2015)

Eine wichtige Frage ist, ob die Assoziationen bidirektional sein sollen. Wir haben uns für die Realisation gemäss der unten stehenden Designentscheidung entschieden.

Themengebiet	Entity-Assoziationen	**Thema**	Design
Name	Uni- oder bidirektionale Assoziationen zwischen Entities	**ID**	DES-DB-ASSOZIATION
Getroffene Entscheidung	Bidirektionale Assoziationen		
Problemstellung	Sollen die Assoziationen zwischen den Entities uni- oder bidirektional designt werden?		
Voraussetzung	Keine		
Motivation	Diese Entscheidung beeinflusst das Persistieren der Models in der Datenbank sowie die Art, wie auf die Informationen der referenzierten Tabellen zugegriffen werden kann.		
Alternativen	• Unidirektionale Assoziationen ○ Vorteil ▪ Bei sehr grossen Datenmengen auf der Many-Seite bei einer OneToMany- oder ManyToOne-Verknüpfung ist eine unidirektionale Assoziation performanter ○ Nachteil ▪ Referenzierte Entities müssen über zusätzliche SQL Queries abgefragt werden		
Begründung	Wir haben uns an der Best-Practice-Empfehlung von Hibernate orientiert und uns entschlossen, alle Assoziationen bidirektional zu machen [14]. Der Vorteil davon ist, dass man aus jeweils beiden Entities heraus ohne zusätzliche Abfragen auf die referenzierten Entities zugreifen kann. Dies erspart uns zusätzliche SQL Queries und vereinfacht die Operationen auf die Models. Auch beim Generieren der JSONs für die HTTP Responses werden durch die bidirektionalen Verknüpfungen die referenzierten Entities automatisch miteinbezogen.		
Annahmen	Der Zugriff auf die referenzierten Entities wird regelmässig benötigt.		
Abgeleitete Anforderungen	Um Dependency Cycles, die mittels der bidirektionalen Assoziationen entstehen, bei der Serialization zu unterbrechen, werden die zusätzlichen Annotationen @JsonManagedReference und @JsonBackReference benötigt[25].		
Verknüpfte Entscheidungen	keine		

Designentscheidung 8 Uni- oder bidirektionale Assoziationen zwischen Entities

Die resultierenden Tabellen sind im nachfolgenden Diagramm mit allen Assoziationen zu sehen. Hibernate hat jeweils auf der ManyToOne-Seite der Assoziationen die Id der One-Seite als ForeignKey eingeführt. Bei OneToOne-Assoziationen wird die Id der PrimaryEntity als ForeignKey in der ChildEntity verwendet (Beispiel Tabellen qavar und valrange). Für die ManyToMany-Assoziationen erstellt Hiberante automatisch, falls nicht via @JoinTable manuell deklariert, eine zusätzliche Tabelle (Beispiel Tabelle qa und qacategory, referenziert über qacategory_qa).

[25] Das Jackson Feature für das Handling von bidirektionalen Referenzen ist unter http://wiki.fasterxml.com/JacksonFeatureBiDirReferences dokumentiert. (zuletzt aufgerufen am 19.05.2015)

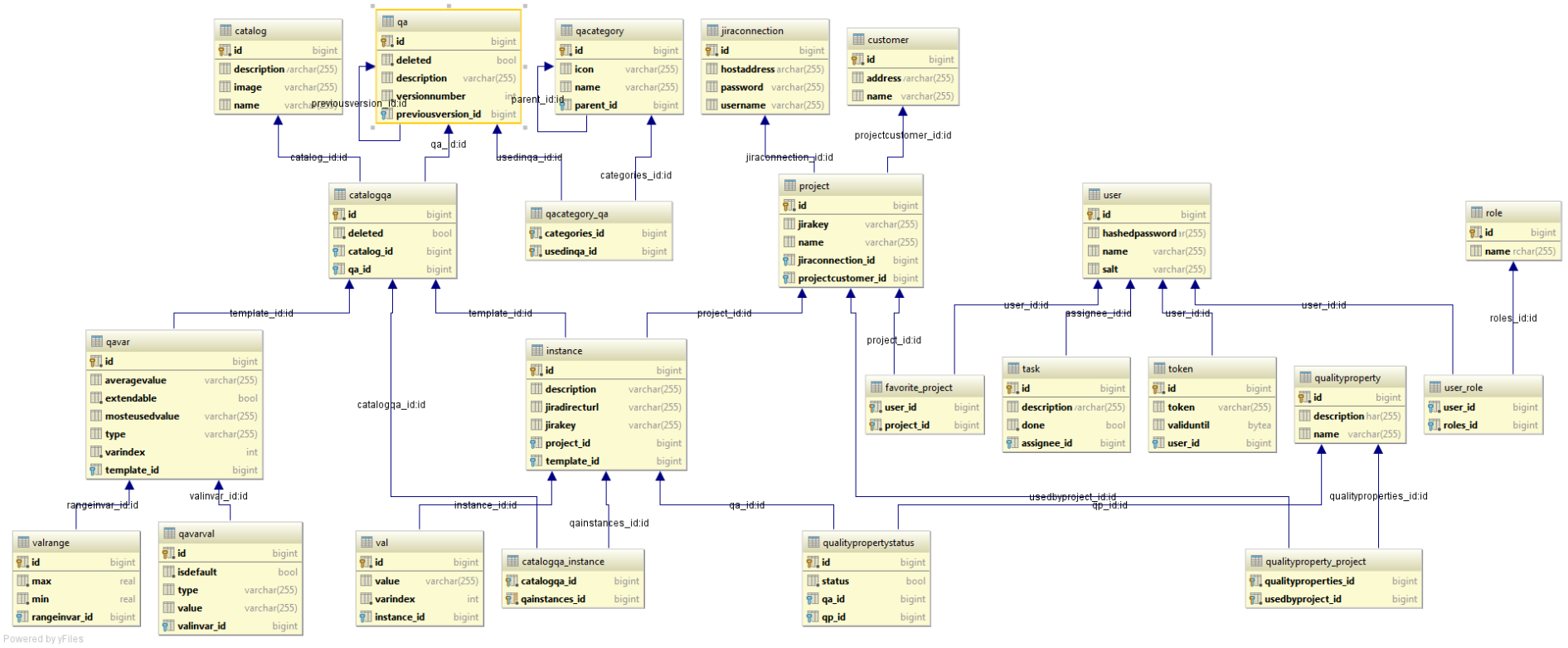


Abbildung 43 Datenbank-Tabellen

Bei Datenbankoperationen auf die Entities gibt es verschiedene Strategien, was mit den referenzierten Entities passieren soll. In unserer Software verwenden wir je nach Bedarf die folgenden drei CASCADE Types:

Cascade-Funktion	Auswirkung
PERSIST	Referenzierte Entities werden bei den persist()-Operationen ebenfalls persistiert. Persist() wird verwendet, um neue Entities anzulegen.
MERGE	Die referenzierten Entities werden bei der update()-Operation mitaktualisiert. Diese Operation wird bei Änderungen auf bereits persistierte Objekte angewendet.
REMOVE	Alle referenzierten Entities werden bei remove() ebenfalls gelöscht.

Tabelle 19 Cascade Types

3.3.3.3.1 Improved Hibernate Dialect

Unit-Tests im Backend testen unter anderem Data Access Objects, welche auf die Datenbank zugreifen. Beim Ausführen eines solchen Tests wird die Datenbank, inkl. Tabellen, gelöscht und neu erstellt. Dieser Schritt erwies sich als problematisch. Die Konsolenausgabe sah folgendermassen aus:

```
[quali-t] $ testOnly unit.daos.template.QualityAttributeDAOTest
[info] Updating {file:/Users/emre/git/quali-t/quali-t-app/}root...
[info] Resolving org.apache.xmlgraphics#batik-anim;1.7 ...
 ...
[info] Resolving org.jacoco#org.jacoco.agent;0.7.1.201405082137 ...
[info] Done updating.
[info] play - datasource [jdbc:h2:mem:play-test--1212769955;MODE=PostgreSQL;DATABASE_TO_UPPER=false] bound
to JNDI as DefaultDS
[error] o.h.t.h.SchemaExport - HHH000389: Unsuccessful: alter table catalogqa drop constraint
FK_h0a9gor6m0r3h6gw0b8ns3b4h if exists
[error] o.h.t.h.SchemaExport - Table "catalogqa" not found; SQL statement:
alter table catalogqa drop constraint FK_h0a9gor6m0r3h6gw0b8ns3b4h if exists [42102-175]
[error] o.h.t.h.SchemaExport - HHH000389: Unsuccessful: alter table catalogqa drop constraint
FK_i0pq178lmjvlovf6wgif2s4iw if exists[error] o.h.t.h.SchemaExport - Table "catalogqa" not found; SQL statement:
```

Code 6 Konsolenausgabe nach Ausführung eines Unit-Tests, welcher auf die Datenbank zugreift (ohne Improved Hibernate Dialect)

Dieses Problem ist bekannt und wird in einem JIRA Ticket[26] des Herstellers als Bug beschrieben. Es existiert aber ein Workaround. Wir haben die Hibernate-Konfiguration (conf/persistence.xml) folgendermassen angepasst:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
        version="2.0">

  <persistence-unit name="defaultPersistenceUnit" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <non-jta-data-source>DefaultDS</non-jta-data-source>
    <properties>
      <property name="hibernate.dialect"
                value="hibernate.dialect.ImprovedPostgreSQL82Dialect"/>
      <property name="hibernate.hbm2ddl.auto" value="create"/>
      <property name="hibernate.hbm2ddl.import_files" value="import.sql"/>
    </properties>
  </persistence-unit>
</persistence>
```

Code 7 Hibernate Konfiguration (persistence.xml) für die Nutzung eines Dialect (Property mit Namen „hibernate.dialect“)

Und die im Ticket erwähnte Klasse in unser Projekt eingefügt:

[26] https://hibernate.atlassian.net/browse/HHH-7002 (zuletzt aufgerufen am 05.06.2015)

```
public class ImprovedPostgreSQL82Dialect extends PostgreSQL82Dialect {
    @Override
    public String getDropSequenceString(String sequenceName) {
        // Adding the "if exists" clause to avoid warnings
        return "drop sequence if exists " + sequenceName;
    }

    @Override
    public boolean dropConstraints() {
        // We don't need to drop constraints before dropping tables, that just
        // leads to error messages about missing tables when we don't have a
        // schema in the database
        return false;
    }
}
```

Code 8 Erweiterung des Postgres-Dialekts, bei welcher die dropConstraints()-Methode überschrieben (bzw. deaktiviert) wird

3.3.3.3.2 Initialer Datenimport

Die Backend-Applikation wird mit Standarddaten (z.B. Roles, Quality Properties und Categories) synchronisiert. Dieser Prozess wird von Hibernate direkt unterstützt. In der Hibernate-Konfiguration (persistence.xml, zu finden im Kapitel 3.3.3.3.1) haben wir den Import mit dem Property „hibernate.hbm2ddl.import_files" definiert.

Für den Import wird die Datei conf/import.sql im Backend gebraucht. Sie ist eine normale SQL-Datei. Wir haben in dieser Datei SQL-INSERT-Befehle definiert. Wir verwenden im import.sql ausschliesslich negative Ids, um Kollisionen mit Daten die von Anwendern erstellt werden zu verhindern.

```
--
-- Data for Name: role; Type: TABLE DATA; Schema: public; Owner: qualit
--

INSERT INTO role VALUES (-20000, 'admin');
INSERT INTO role VALUES (-20001, 'curator');
INSERT INTO role VALUES (-20002, 'analyst');
INSERT INTO role VALUES (-20003, 'synthesizer');
INSERT INTO role VALUES (-20004, 'evaluator');
INSERT INTO role VALUES (-20005, 'projectmanager');
```

Code 9 Auszug aus der import.sql-Datei mit SQL-INSERT-Befehlen für die Initialisierung der Rollen

3.3.3.4 Dependency Injection

In diesem Abschnitt haben wir das im Backend für Dependency Injection eingesetzte Google Guice detailliert dokumentiert. Ausserdem beschreiben wir die Probleme, welche während der Evaluation des Frameworks aufgetreten sind.

3.3.3.4.1 Probleme mit Spring

Spring lässt sich einfach und schnell ins Play Framework integrieren. Jedoch gibt es ein grosses Problem. Es handelt sich dabei um das Transaktionsmanagement.

Wir haben die Transaktionsgrenzen mittels @Transactional(play.db.jpa.Transactional)-Annotation in Controller-Methoden festgelegt.

```
@Transactional
public Result getAllQAs() {
    return ok(Json.toJson(qualityAttributeLogic.getAllQAs()));
}
```

Code 10 Transaktionsgrenze wird in der Controller-Methode mit der Annotation @Transactional gesetzt

Spring hingegen erwartet die Annotation aus dem eigenen Framework (org.springframework.transaction.annotation.Transactional). Diese Annotation wird aber vom Play Framework nicht unterstützt und sobald innerhalb der Controller-Methode auf die Datenbank zugegriffen wird, wird eine Exception geworfen, dass kein Entity Manager im aktuellen Thread gefunden wurde.

3.3.3.4.2 Beispiel mit Google Guice

Google Guice lässt sich gemäss offizieller Dokumentation[27] einfach integrieren. Danach kann Google Guice gebraucht werden.

Die zu injizierenden Objekte werden mit @Inject annotiert. Sofern keine zusätzlichen Einstellungen (z.B. Singleton) benötigt werden, reicht dies aus.

```
public class QualityAttributeController extends Controller implements ExceptionHandlingInterface {

    @Inject
    private QualityAttributeLogic qualityAttributeLogic;

    // ausgeblendeter Code

    @Transactional
    public Result getAllQAs() {
        return ok(Json.toJson(qualityAttributeLogic.getAllQAs()));
    }
}
```

Code 11 Injizierung des Objekts qualityAttributeLogic

3.3.3.4.3 Unit-Tests

Wir konnten innerhalb der Unit-Tests im Backend ebenso die Dependency-Injection-Funktionalität ausnutzen. Da für die Unit-Tests die Global.java-Klasse ignoriert wird, aber die Konfiguration von Google Guice in dieser Klasse gemacht wird, mussten wir einen Workaround implementieren.

Wir haben eine AbstractTest-Klasse eingeführt. Jede Testklasse, welche Dependency Injection braucht, muss diese Klasse erweitern.

```
public abstract class AbstractTest {
    private Injector injector = TestDependencyUtil.createInjector();

    public Injector getInjector() {
        return injector;
    }
}
```

Code 12 Google-Guice-Konfiguration in der abstrakten Testklasse

Im Gegensatz zum Applikations-Code müssen die Objekte in der Testklasse zusätzlich manuell injiziert werden.

[27] https://www.playframework.com/documentation/2.3.x/JavaInjection (zuletzt aufgerufen am 05.06.2015)

```
public class AuthenticationLogicTest extends AbstractTest {
    @Inject
    Authenticator authenticator;

    @Before
    public void setUp() throws Exception {
        authenticator = getInjector().getInstance(Authenticator.class);
    }

    @Test
    public void isTokenValidWithValidTokenTest() {
        LocalDateTime validUntil = LocalDateTime.now().plusDays(30);
        Token t = new Token("xxx", validUntil, null);
        assertThat(authenticator.isTokenValid(t)).isTrue();
    }
}
```

Code 13 Eine Testklasse, welche die abstrakte Klasse AbstractTest erweitert, um Dependency-Injection-Funktionalität zu erhalten

3.3.3.5 API Dokumentation

In diesem Abschnitt ist unsere API dokumentiert. Da die komplette Dokumentation sehr umfangreich ist, wird an dieser Stelle nur ein Auszug gezeigt. Der Auszug beinhaltet die allgemeinen Informationen sowie die möglichen HTTP-Methoden auf den ProjectInitiator. Die komplette Dokumentation ist im Anhang verfügbar.

3.3.3.5.1 Struktur der Uniform Resource Identifiers (URIs)

QUALI-T bietet Zugang zu allen Daten und CRUD Funktionen direkt über URI Pfade, welche von der Backend API zur Verfügung gestellt werden. Der Aufbau der URI setzt sich aus einem **statischen** Teil zusammen gefolgt von dem Ressourcentyp, die bearbeitet werden soll. Wird ein GET auf ein explizites Objekt gemacht, so wird die Id direkt in der URI übermittelt. Bei allen anderen HTTP Verben wird die ID innerhalb des Bodys übermittelt. Der Body besteht bei allen POST/PUT Requests und Responses aus einem JSON. Bei GET und DELETE Requests wird kein Body verwendet.

http://host:port/api/{ressourcename}/*id*

Beispiel für den Zugriff auf alle Projekte über Heroku:
http://quali-t.herokuapp.com/api/project

Beispiel für den Zugriff auf das Projekt mit der Id -11000:
http://quali-t.herokuapp.com/api/project/-11000

3.3.3.5.2 Authentifizierung

Alle URIs, ausser diejenigen, welche für die Authentifizierung gebraucht werden, sind geschützt. Wie man sich authentifiziert ist im Kapitel **Error! Reference source not found.** detailliert beschrieben.

3.3.3.5.3 Beispiel Project Initiator

Im nachfolgenden ist die Dokumentation des ProjectInitiator-Objekts als Beispiel gezeigt.

3.3.3.5.3.1 api/projectInitiator

PUT

Updated einen bestehenden ProjectInitiator;

Request:

- application/json

```
{
  id: 61,
  name: "Farbmuster AG",
  address: "Schaffhausen-Nord"
}
```

Responses:

- 400

Id ist ungültig;

- 422

name ist bereits vorhanden;
name fehlt, ist *null*;
id fehlt, ist *null*;

- 200 - application/json

analog zu der 200 POST Response von **Error! Reference source not found.**

3.3.3.5.3.2 api/projectInitiator/{id}

DELETE

Löscht einen ProjectInitiator;

Request:

- Parameter {id}

Id des ProjectInitiators, welcher gelöscht werden soll.

Responses:

- 400

Id ist ungültig;

- 422

id fehlt, ist *null*;

- 204

3.3.4 User Management

Die Wichtigkeit des User Management ist in der Bachelorarbeit mit einer tieferen Priorität eingestuft. Da QUALI-T aus zwei Applikationen (Frontend und Backend) besteht, müssen beide ein Konzept für User Management implementieren.

Die Authentifizierung (z.B. Login, Benutzer anlegen) haben wir selbst entwickelt. Die detaillierte Beschreibung ist in diesem Kapitel enthalten.

3.3.4.1 User Management im Backend

Die Backend-Applikation bietet eine RESTful HTTP API (Kapitel 3.1.10) an, welche schützenswert ist. Für die Authorization haben wir das Play Framework Modul Deadbolt[28] verwendet. Auf die Authentifizierung wird nicht eingegangen (da zu simpel).

3.3.4.1.1 QualitDeadboltHandler

Deadbolt lässt sich einfach integrieren. Wir konnten für QUALI-T einen sogenannten DeadboltHandler implementieren. Diese Klasse enthält das Verhalten für die Autorisierung.

```
public class QualitDeadboltHandler extends AbstractDeadboltHandler {
    @Override
    public F.Promise<Result> beforeAuthCheck(Http.Context context) {
        return F.Promise.pure(null);
    }

    @Override
    public Subject getSubject(Http.Context context) {
        try {
            return JPA.withTransaction(new play.libs.F.Function0<User>() {
                public User apply() {
                    // TODO emre: move to userid instead of username -> easier
                    String u = context.session().get("username");
                    UserDao userDao = new UserDao();
                    User loggedInUser = userDao.findByUsername(u);
                    return loggedInUser;
                }
            });
        } catch (Throwable throwable) {
            play.Logger.error("Error at accessing the database.");
        }

        return null;
    }
}
```

Code 14 Implementation des QualitDeadboltHandler für die Autorisierung

Die QualitDeadboltHandler-Klasse muss konfiguriert (in application.conf) werden.

```
# Deadbolt Plugin
deadbolt.java.handler= logics.authentication.QualitDeadboltHandler
```

Code 15 Konfiguration von QualitDeadboltHandler für die Autorisierung in der Backend-Applikation

3.3.4.1.2 Absicherung der Controller

Die Controller-Klassen bieten einen Eingang in die Backend-Applikation. Die Methoden in diesen Klassen müssen also gesichert werden.

Deadbolt ermöglicht dies durch die Verwendung von Annotationen. Im unten stehenden Beispiel ist die Methode createQA (in der QualityAttributeController-Klasse) nur durch Benutzer, welche die Rolle „curator“ oder „admin“ haben, aufrufbar.

```
@Restrict({@Group("curator"), @Group("admin")})
@Transactional
public Result createQA() {
    // implementation
}
```

Code 16 Annotation einer Controller-Methode für die Autorisierung in der Backend-Applikation

[28] http://deadbolt.ws (zuletzt aufgerufen am 07.06.2015)

3.3.4.2 User Management in der Frontend-Applikation

Da die Frontend-Applikation clientseitig ist, könnte sie vom Benutzer verändert werden. Deshalb ist das User Management in der Frontend-Applikation sehr einfach implementiert.

3.3.4.2.1 Autorisierung

Da ein Benutzer in der Backend-Applikation nicht auf alle Funktionen zugreifen kann, ist es sinnvoll, den Zugang zu diesen Funktionen in der Frontend-Applikation einzuschränken. Die Einschränkungen werden nur auf der Ebene der View (bzw. State in "ui-router"-Modul) gemacht.

Eine Route im Frontend hat die Option, mit Rollen, die auf diese Route Zugriff haben, erweitert zu werden.

```
angular.module('qualitApp')
  .config(function ($stateProvider, $urlRouterProvider) {
    $urlRouterProvider.otherwise('/welcome');

    $stateProvider
        .state('newCatalog', {
            templateUrl: 'views/catalog/new.html',
            url: '/authenticated/catalog/create',
            controller: 'CreateCatalogCtrl',
            data: {
              roles: ['curator', 'admin']
            }
          });

    // andere states
});
```

Code 17 Restriktion einer Route in der Frontend-Applikation, sodass nur Catalog Curators und Admins Zugriff haben

3.3.5 Import / Export von Daten

In diesem Kapitel sind alle Schnittstellen zu anderen Systemen sowie Import- und Export-Funktionen dokumentiert.

3.3.5.1 JIRA Project Issue Tracking System

QUALI-T bietet die Möglichkeit, Projekt-QA-Instances direkt in ein JIRA-Projekt zu exportieren. Der Export des QA kann entweder als Raw-Text oder mit den im QUALI-T verwendeten Hypertext Markup Language (HTML) Tags erfolgen.

3.3.5.1.1 JIRA-Export in QUALI-T

Der Export erfolgt jeweils an eine vordefinierte JIRA-Instanz in ein bereits bestehendes JIRA-Projekt. Dazu sind die Export-Properties *„JIRA Instance“* und *„JIRA Key“* notwendig.

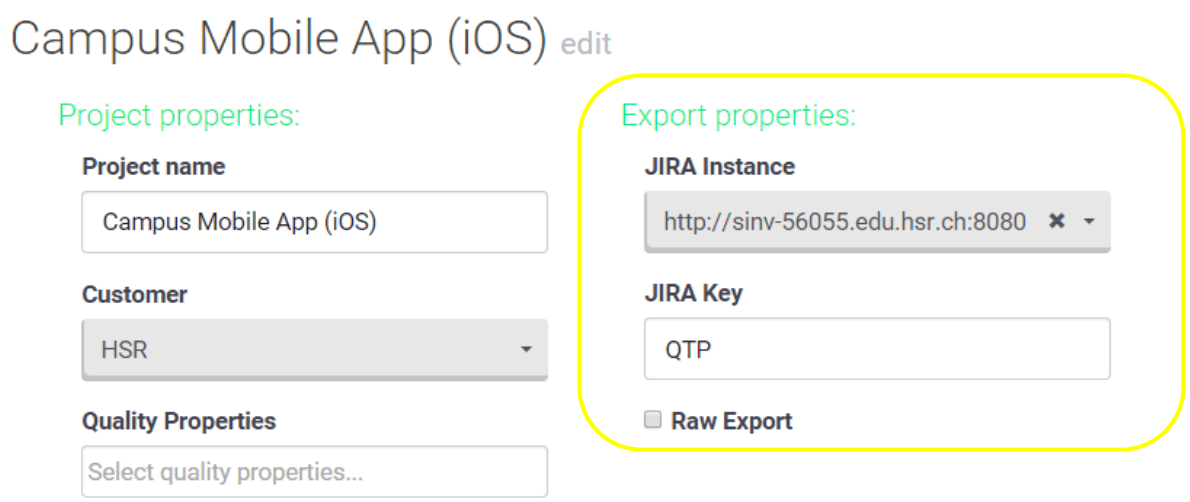


Abbildung 44 JIRA Instance und JIRA Key

Anschliessend werden die gewünschten QA Instances – und wenn nötig der *„Raw Export“*-Button – ausgewählt und über den *„Export to issue tracking system“*-Button exportiert.

Bei erfolgreichem Export wird unterhalb des QA-Texts direkt ein Link zur JIRA Issue publiziert.

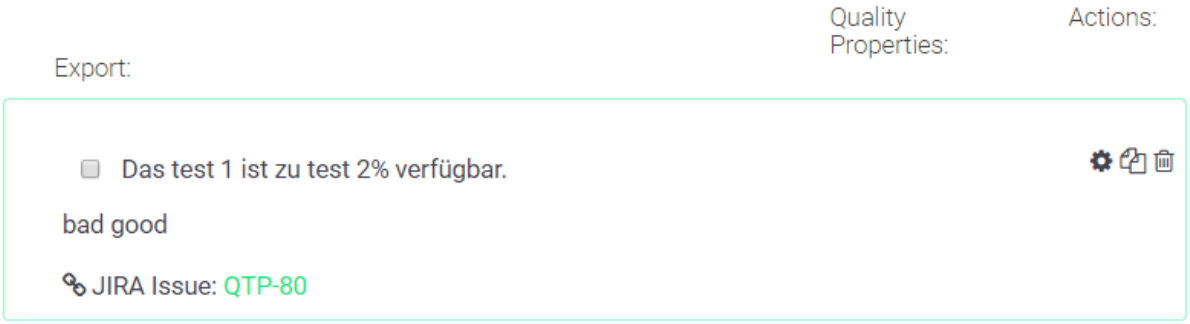


Abbildung 45 QA Instance mit JIRA-Issue-Link

3.3.5.1.2 JIRA-Schnittstelle

Die ausgewählten QA Instances werden im Backend mit den Variablen aufbereitet, sodass der angezeigte Text lesbar ist. Anschliessend wird versucht, über einen POST pro QA Instance eine Issue zu erstellen. Existiert bereits eine Issue, so wird ein Kommentar zur bestehenden Issue hinzugefügt. Der Kommentar beinhaltet den aktuellen Text.

Der HTTP-Aufruf sowie die verwendeten Parameter lassen sich den nachfolgenden Tabellen entnehmen.

Funktion	HTTP Verb	JIRA API Address	Message Body [JSON]

Issue erstellen	POST	{JIRAHostAddress}/rest/api/latest/issue	{
			"fields": {
			"project": {
			"key": "{JIRAProjectKey}"
			},
			"summary": "Project {ProjectName} / QA {QAId}",
			"description": "{QAText}",
			"issuetype": {
			"name": "Task"
			}
			}
			}

Issue aktualisieren (Kommentar erstellen)	POST	`{JIRAHostAddress}/rest/api/latest/issue/{JIRA-Key}/comment`	{
			"body": "{QAText}"
			}

Tabelle 20 JIRA-Export-Schnittstelle

Parameter	Erklärung	Beispiel
{JIRAHostAddress}	Host-Adresse der JIRA-Instanz, wird dem JIRA Instance Attribute entnommen.	http:// sinv-56055.edu.hsr.ch:8080
{JIRAIssueKey}	JIRA Issue Key der QA Instance; wird nach erfolgreichem Erstellen einer Issue als Attribut zum QA-Instance-Objekt hinzugefügt.	QTP-80
{QAText}	Text kann entweder im Raw-Format oder mit den im QUALI-T verwendeten HTML Tags vorliegen.	Das System ist zu 99,5% verfügbar.
{JIRAProjectKey}	JIRA Key des Projekts, in welchem die Issues angelegt werden	QTP
{ProjectName}	Name des Projekts in QUALI-T	Campus Mobile App (iOS)
{QAId}	ID der QA Instance in QUALI-T	89

Tabelle 21 JIRA Export Parameter

3.3.5.2 Import und Export von Catalogs

Ein kompletter Catalog kann als JSON exportiert werden. Der Export beinhaltet alle Catalog QAs sowie die zugehörigen Variable Values und Variable Ranges (falls vorhanden).

Beim Import wird der komplette Catalog mit allen notwendigen QAs, CatalogQAs und dazugehörigen Variablen angelegt. Die importierten CatalogQAs sind ebenfalls im Standard-Catalog mit den gleichen Variablen wie im importierten Catalog verfügbar. Die QAs werden zu den entsprechenden QA Categories hinzugefügt, falls im Ziel QUALI-T bereits eine QA Category mit dem gleichen Namen existiert. Ansonsten werden die QA Categories beim Import ignoriert.

3.3.5.3 Backup und Restore QUALI-T-Instanz

Um alle Daten einer QUALI-T-Instanz zu sichern, kann ein SQL Dump mit Postgres erstellt werden[29]. Dieser kann später auch wieder 1:1 in eine bestehende Datenbank importiert werden. Da bei einem Dump die komplette Datenbank geleert respektive neu befüllt wird, können hier auch alle bestehenden IDs wiederverwendet werden.

Um einen Dump mit dem Uster *postgres* unter C:\Temp zu erstellen, werden folgende Befehle in einer Shell eingegeben:

```
cd C:\Program Files\PostgreSQL\9.4\bin
pg_dump -d qualit -h localhost -U postgres > C:\Temp\qualit_dump.backup
```

Code 18 Erstellen eines SQL Dump

Das Backupfile kann anschliessend in eine leere Datenbank importiert werden. Diese muss den Namen *qualit* haben.

```
cd C:\Program Files\PostgreSQL\9.4\bin
psql -d qualit -h localhost -U postgres < C:\Temp\qualit_dump.backup
```

Code 19 Importieren eines SQL Dump

3.3.5.4 Project Report

QUALI-T bietet die Möglichkeit, alle Quality Attributes eines Projekts als Report zu exportieren. Dabei kann zwischen XML und PDF gewählt werden. Der Export als XML bietet eine einfache Möglichkeit, die Daten in anderen Programmen wiederzuverwenden.

3.3.5.4.1 Export in QUALI-T

In der „Edit Project“-View kann via Button das gesamte Projekt im gewünschten Format exportiert werden:

[29] Weitere Informationen zum erstellen von Dumps in Postgres sind unter http://www.postgresql.org/docs/9.4/static/backup-dump.html verfügbar (zuletzt aufgerufen am 10.05.2015)

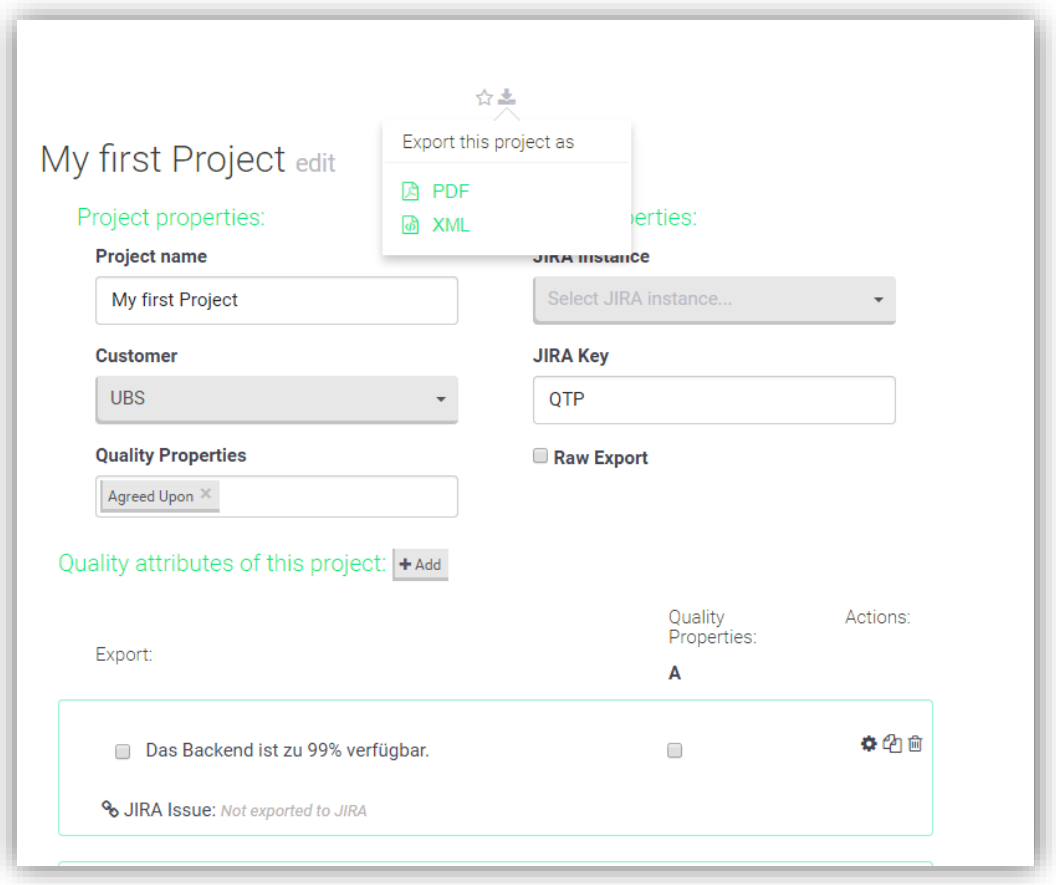


Abbildung 46 Export Project als XML oder PDF

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <project jiraKey="QTP" name="My first Project">
    <projectCustomer name="UBS" address="Zürich"/>
   - <qualityAttributes>
      <qualityAttribute description="Das GUI muss in Englisch verfügbar sein."/>
      <qualityAttribute description="Das Backend ist zu 99% verfügbar."/>
    </qualityAttributes>
  </project>
```

Abbildung 47 XML-Exportbeispiel

Abbildung 48 PDF-Exportbeispiel

3.3.5.4.2 Export mit Apache FOB[30]

Der Export ist mit Apache FOB realisiert. Dazu werden zuerst die speziell für den XML-Export angelegten Java-Model-Objekte mit einem dafür geschriebenen Converter mit Daten befüllt. Es werden nicht die normalen Models verwendet, da nicht alle Daten für den Export relevant sind und spezielle Annotationen verwendet werden, die ausserhalb der Exportfunktion keinen Nutzen haben. Die Annotationen gehören zu dem Oracle Java Package javax.xml.bind.annotation[31]. Mit Hilfe dieser Annotationen generiert QUALI-T mit Apache FOB anschliessend ein XML-File wie in Abbildung 47 XML-Exportbeispiel gezeigt.

Wählt man als Exportformat PDF, wird zuerst ebenfalls das XML generiert. Anschliessend benötigt QUALI-T zusätzlich ein XSL-File. Im XSL-File ist beschrieben, wie das PDF aufgebaut ist, wo welche Daten angezeigt werden und wie die Formatierung aussieht. Eine detaillierte Beschreibung des verwendeten XSL-FO ist unter http://www.w3.org/TR/xsl/ verfügbar.

3.3.5.4.2.1 Bekannte Compiler Warnings

Beim Compilen von QUALI-T tritt eine Warnung auf. Diese stellt jedoch kein Problem dar und gehört zu einem bekannten Bug, welcher bei Oracle gelistet ist[32]. Ab Version 8 Build96 ist dieser Bug jedoch behoben. Die Warnungen erscheinen nicht mehr als Fehler und nicht mehr in roter Schrift.

3.3.6 Lizenzen und verwendete externe Produkte

Alle verwendeten Lizenzen und externe Produkte sind im Anhang zu finden.

3.4 Testing

Gute Code-Qualität erfordert Testing. In diesem Kapitel gehen wir auf die verwendeten Testmethoden ein und erklären sie in detaillierter Form. Die Tests haben wir gemäss Praxiserfahrung und Internetrecherche [15] folgendermassen aufgeteilt:

[30] Apache FOB ist ein Appache-Projekt zum Erstellen von verschiedenen Ausgabeformaten mittels XSL-FO.
[31] Die javax.xml.bin.annotation-Annotations sind unter http://docs.oracle.com/javase/8/docs/api/javax/xml/bind/annotation/package-summary.html dokumentiert. (zuletzt aufgerufen am 05.05.2015)
[32] http://bugs.java.com/view_bug.do?bug_id=8016153 (zuletzt aufgerufen am 01.05.2015)

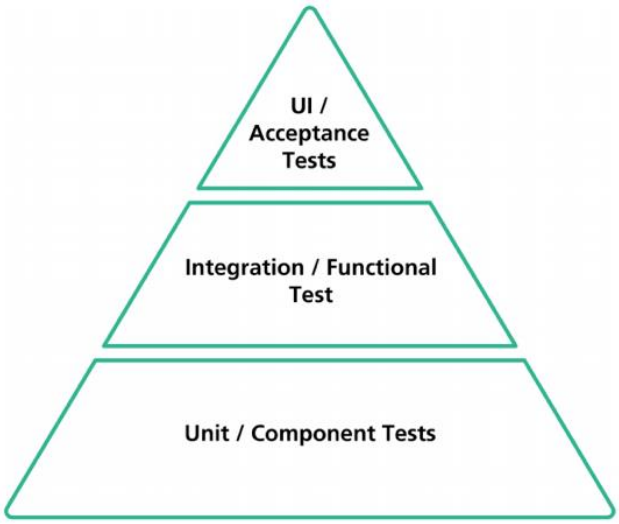


Abbildung 49 Testing Pyramide mit der Aufteilung der Testmethoden (Präsentation Essentials of Testing, Jazoon 2010)

3.4.1 Unit-Tests

Die Qualität der Backend-Applikation wird mittels Unit-Test überprüft. Dazu haben wir JUnit-Tests für die Public-Methoden der Logic-Klassen sowie für die speziellen Methoden der DAOs geschrieben. Zu den speziellen Methoden gehören alle, die nicht in der *AbstractDAO*-Klasse enthalten sind. Auf das Testen der Controller- und Model-Klassen haben wir bewusst verzichtet, da diese keine Logik beinhalten und die Controller teilweise schon in den Integration Tests miteinbezogen sind. Wir haben daher keine 100% Test Coverage angestrebt, sondern uns auf das Testen der Kernfunktionen beschränkt. Aus diesem Grund werden auch reine Datenabfrage-Methoden nicht getestet.

```
cd quali-t-app

# führt alle Tests aus
activator test
```

Code 20 Befehle für die Ausführung der Unit-Tests im Backend

3.4.1.1 Test Coverage mit JaCoCo

Eine detaillierte Test-Coverage-Statistik wird mit Hilfe des Open-Source-Tools jacoco4sbt (Code Coverage via JaCoCo in sbt) erstellt. Dies ist ein Typesafe Activator und sbt Plugin für die Code-Coverage-Analyse mit JaCoCo.[33]

JaCoCo kann mittels folgendem Befehl in einem Konsolenfenster ausgeführt werden. Zuerst muss allerdings ins Runtime-Verzeichnis der QUALI-T-App gewechselt werden:

```
cd c:\ba\git\quali-t\quali-t-app
activator test jacoco:check
```

Code 21 JaCoCo Code Coverage

Mit diesem Befehl sind die Coverage-Reporte auch verfügbar, wenn die Applikation nicht läuft. Der erstellte HTML-Report ist im QUALI-T-App-Ordner über C:/BA/GIT/quali-t/quali-t-app/target/scala-2.11/jacoco/html/index.html aufrufbar.

[33] Jacoco4sbt ist unter https://github.com/sbt/jacoco4sbt verfügbar und dokumentiert. (zuletzt aufgerufen am 02.06.2015)

Nach der Ausführung in der Kommandozeile erscheint auch dort direkt eine Übersicht der Code Coverage, diese ist in der folgenden Abbildung gezeigt:

```
[info] ------- QUALI-T Code Coverage Report --------
[info]
[info] Lines: 68.41% (>= required 0.0%) covered, 581 of 1839 missed, OK
[info] Instructions: 69.56% (>= required 0.0%) covered, 2458 of 8076 missed, OK
[info] Branches: 64.49% (>= required 0.0%) covered, 223 of 628 missed, OK
[info] Methods: 67.55% (>= required 0.0%) covered, 147 of 453 missed, OK
[info] Complexity: 62.08% (>= required 0.0%) covered, 292 of 770 missed, OK
[info] Class: 80.25% (>= required 0.0%) covered, 16 of 81 missed, OK
[info] Check C:\BA\GIT\quali-t\quali-t-app\target\scala-2.11\jacoco for detail report
[info]
```

Abbildung 50 QUALI-T Code Coverage Übersicht

Die nachfolgende Abbildung zeigt die zum BA-Abgabetermin aktuelle Code Coverage von QUALI-T im HTML-Report mit allen berücksichtigten Packages. Dieser Report kann mit den oben genannten Instruktionen jederzeit interaktiv generiert werden.

QUALI-T Code Coverage Report

QUALI-T Code Coverage Report

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
dao		91%		80%	4	17	4	35	2	12	0	1
dao.authentication		60%		50%	2	7	4	11	1	5	0	2
dao.interfaces		100%		n/a	0	1	0	1	0	1	0	1
dao.models		82%		44%	7	39	15	69	2	31	1	14
dao.user		0%		n/a	1	1	1	1	1	1	1	1
default		74%		n/a	1	4	1	7	1	4	0	1
exceptions		71%		n/a	3	10	6	21	3	10	3	10
hibernate.dialect		100%		n/a	0	3	0	3	0	3	0	1
logics.authentication		78%		69%	22	49	23	101	6	18	1	3
logics.interfaces		30%		48%	17	30	41	57	6	10	1	2
logics.interfaces.projectExport.models		2%		0%	19	20	51	52	17	18	3	4
logics.interfaces.projectExport.repositories		7%		n/a	3	5	19	21	3	5	0	2
logics.project		80%		77%	36	110	50	243	8	33	1	4
logics.search		100%		n/a	0	6	0	15	0	6	0	1
logics.template		87%		83%	23	93	31	222	8	33	0	3
logics.user		0%		0%	35	35	60	60	13	13	3	3
logics.validation		0%		0%	20	20	53	53	5	5	2	2
models		41%		n/a	1	3	3	6	1	3	0	1
models.authentication		62%		0%	16	32	34	86	14	30	0	3
models.Interface		60%		n/a	2	7	8	22	2	7	0	1
models.project		46%		50%	16	36	51	105	13	32	0	3
models.project.nfritem		66%		58%	11	42	38	125	8	36	0	3
models.template		78%		93%	27	102	64	271	25	87	0	9
models.user		7%		0%	6	7	13	15	5	6	0	1
util		95%		78%	20	91	11	237	3	44	0	5
Total	2,458 of 8,076	70%	223 of 628	64%	292	770	581	1,839	147	453	16	81

Abbildung 51 QUALI-T Code Coverage Report mit jacoco4sbt

In den beiden obigen Abbildungen ist zu sehen, dass wir eine Code Coverage von etwa 70% erreichen. Aus den am Anfang im Kapitel erwähnten Gründen liegt die Coverage nicht höher, aber deckt die Kernfunktionen ab und ist somit ausreichend.

3.4.2 Integrationstests

Die Integrationstests haben wir in der Frontend-Applikation realisiert. Hierfür haben wir das Testing Framework Protractor benutzt. Dieses Framework wird von AngularJS empfohlen und wurde vom AngularJS Team entwickelt.

Die Tests sind vergleichbar mit Selenium Tests, denn Protractor ist im Prinzip ein Wrapper für die JavaScript Selenium Webdriver[34]. Für die Tests werden die Browser Firefox und Chrome in der aktuellen Version benötigt. Beim Ausführen der Tests werden dann die Browser gestartet und die Integrationstests durchgeführt.

Die Tests können mit folgendem Befehl (in der Konsole) ausgeführt werden:

```
cd quali-t-app/public

# führt alle Tests (falls z.B. Unit Tests vorhanden) aus
grunt test

# führt alle protractor Tests aus
grunt protractor

# führt alle protractor Tests mit dem Chrome Browser aus
grunt protractor:chrome
```

Code 22 Befehle für die Ausführung der Frontend-Tests

Integrationstests ergänzen die Unit-Tests aus dem Backend. Deshalb haben wir die wichtigsten Use Cases (UC1 – CRUDS Projekt-Metadaten, UC2 – CRUDS NFR Template Set, UC3 – CRUDS NFR Item Template, UC4 – Login) mit Integrationstests abgedeckt. Zusätzlich zu den Use Cases sind noch weitere Integrationstests (z.B. Erstellung eines Accounts) implementiert.

[34] http://webdriver.io (zuletzt aufgerufen am 08.06.2015)

3.4.3 Usability-Tests

Zum Abschluss der Testreihe haben wir auch einen Usability-Test durchgeführt. Primär richtet sich dieser an unseren BA-Betreuer. Der Test wurde aber auch von einigen neutralen Personen durchgeführt. Der Test ist online[35] verfügbar und befindet sich als Print-Version im Anhang.

3.4.3.1 Usability-Test mit BA-Betreuer

Herr Zimmermann hat für diesen Test die am 25.06.2015 aktuelle QUALI-T-Version unter http://dev.quali-t.ch verwendet. Folgendes Feedback haben wir erhalten:

Nr	Frage	Antwort
	Timestamp	01/06/15
1	Bitte öffnen sie die Webapplikation QUALI-T (http://dev.quali-t.ch). Uns interessiert der erste Eindruck. Wie finden Sie sich auf der Hauptseite zurecht?	gut
2	Sie können nun in die Web-Applikation einloggen. Zum Login-Bereich können Sie via Klick auf "Login" Button auf der Hauptseite gelangen. Als Benutzer können Sie admin (mit Passwort admin) verwenden.	Ich konnte mich einloggen.
3	Sie werden nun einige Szenarien durchspielen. Bevor Sie dies tun, möchten wir, dass Sie selbst die Applikation kennenlernen. Ziel ist es die allgemeine Nutzbarkeit zu bewerten und vor allem unerklärliches Verhalten der Applikation ausfindig zu machen. Wie ist Ihr Eindruck? (Bitte löschen Sie nichts)	Beim Einloggen nach Erstellen eines eigenen Accounts oben rechts Fehlermeldung "Could not load Favorites". Bedeutung und Auswirkung CC, AA, AS, AE nicht intuitiv klar für Newbie.
4	Lesen Sie die "Help" Seite. Verstehen Sie den Inhalt?	Ja, ich habe den Sinn der "Key Elements" (z.B. Quality Attribute Template) verstanden.
5	In FAQ (weiterhin auf der "Help" Seite) wird u.A. ein Use-Case beschrieben. Können Sie diesen nachvollziehen? Empfinden Sie diesen Prozess als sinnvoll?	Ja
6	Sie werden nun diesen Use-Case durchspielen. Stellen Sie sich folgendes Szenario vor: In Ihrer Firma wird QUALI-T verwendet. Sie	Überraschend einfach, ich habe das neue Projekt innerhalb von 5 Minuten angelegt.

[35] https://docs.google.com/forms/d/18eVVIpaLNQTfKCSCqdrwnJrT0ME0l-GDr2BBdq3yGNQ (zuletzt aufgerufen am 10.06.2015)

	haben einen neuen Kunden (Name: Muster AG, Adresse: 8005 Zürich) und somit ein neues Projekt gewonnen. Es geht im Projekt um eine Social Learning App für das iOS Betriebssystem. Ihnen wurde gesagt, dass bereits eine iOS App letztes Quartal spezifiziert wurde. Ihre Aufgabe ist es nun, das Projekt in QUALI-T anzulegen. Wie einfach lässt sich das Szenario abdecken?	
7	Falls Sie die vorherige Frage mit "schlecht" oder "sehr schlecht" beantwortet haben, möchten wir Sie gerne fragen, was wir verbessern können und wo die Problemstellen waren.	Testsetup: es ist unklar, welchen QAT Katalog ich für das neue Projekt bei Muster-AG verwenden soll.
8	Sie sollten nun ein Projekt angelegt haben. Sie sollen nun die einzelnen Quality Attributes spezifizieren. Jedes Quality Attribute kann editiert werden (Zahnrad-Icon). Ändern Sie die Werte der Variablen und speichern Sie das Projekt. Waren Sie erfolgreich?	Nein
9	Ihnen ist vielleicht aufgefallen, dass Sie neue Quality Attributes zum bestehenden Projekt hinzufügen können. Probieren Sie diese Funktionalität aus. Waren Sie erfolgreich?	Ja
10	Sie haben entschieden, dass die erfassten Quality Attributes im Projekt gewisse Ziele / Eigenschaften erfüllen sollen. Zum Beispiel soll ein Quality Attribute messbar und spezifisch sein. Fügen Sie diese Eigenschaften (Quality Properties) dem Projekt hinzu. Sehen Sie die Quality Properties pro Quality Attribute (Checkbox)?	Ja
11	Da Sie nun die Quality Attributes spezifisch und messbar gemacht haben, können Sie nun die Quality Attributes bewerten. Bitte bewerten Sie die ersten 3 Quality Attributes und setzen Sie den richtigen Wert pro Checkbox. Wie einfach gestaltet sich dieser Vorgang?	Normal, entspricht den Anforderungen.
12	Sie können die jeweiligen Quality Attributes in ein externes Projektmanagement System (JIRA)	Nein

	exportieren. Probieren Sie es aus. Hat es funktioniert?	
13	Vielleicht haben Sie den Button "Validate" bemerkt. Wenn Sie mit der Maus über diesen Button fahren bekommen Sie mehr Informationen (dies funktioniert ausserdem bei vielen Elementen innerhalb der Applikation). Probieren Sie die validate Funktionalität aus. Verstehen Sie das Resultat?	Nein
14	Das nächste Szenario ist ein ähnliches. Der gleiche Kunde möchte nun die App für die Apple Watch lancieren. Bei der Apple Watch sind aber ganz andere Anforderungen wichtig. Sie entscheiden sich, dass Sie die Apple Watch Anforderungen spezifizieren (denn die jetzigen Quality Attribute Templates sind nicht passend). Danach möchten Sie diese Quality Attribute Templates in einen neuen Katalog (Name: App for Apple Watch) speichern. Erstellen Sie also 5 Quality Attribute Templates inkl. Variablen (der Inhalt ist nicht relevant). Erstellen Sie dann einen Katalog mit den Quality Properties S, M, A, R, T. Bei der Auswahl von Quality Attribute Templates können Sie zusätzlich noch ein eigenes (noch nicht vorhandenes) Quality Attribute Template erstellen. Erstellen Sie diese mit dem Inhalt "The app on the Apple Watch has a good user interface". Speichern Sie den Katalog ab. Waren Sie erfolgreich?	Ja, aber ich muss noch ein bisschen üben.
15	Falls Sie die letzte Frage mit "nein" beantwortet haben, bitten wir Sie die Probleme unten zu schildern. Sie dürfen ebenfalls Verbesserungsvorschläge anbringen.	Test nicht mehr durchgeführt wg. diverser Robustheitsprobleme und Fehler beim freien Testen mit eigenen Daten und beim Test im Muster AG Social Learning App Scenario. Ins. Update- und Reloadverhalten. UI Layout bis auf kleinere Probleme mit Formulierungen und Positionierung von Buttons/Icons i.O. Aber aktuell ist ein produktives Arbeiten mit der AW aufgrund der fehlenden Stabilität noch nicht möglich => Fokus auf Bug Fixing und Testing

		Siehe Kommentare und Bug Reports weiter unten.
16	Der Katalog ist nun erstellt. Erstellen Sie das dazugehörige "Social Learning Apple Watch App" Projekt. Nach dem Erstellen des Projekts sollen Sie die Variablen ausfüllen. Klicken Sie nachdem Sie das Projekt erneut gespeichert haben auf validate. Nun sollten bei einigen Quality Attributes ein neues Warn-Icon angezeigt werden. Klicken Sie auf diese. Sind diese Informationen sinnvoll? Können Sie sich vorstellen, dass Sie so besser spezifizieren können?	Welche Warnungen? Ich verstehe es nicht.
17	Sie haben nun alle Szenarien durchgetestet. Wir bitten Sie nun die Kataloge und Projekte jeweils als JSON bzw. PDF und XML zu exportieren und an emre.avsar92@gmail.com zu senden. Hat der Export geklappt?	Nein
18	Wir bitten Sie die Applikation vor Ende des UI-Reviews noch einmal durchzutesten. Da dieser Schritt nun generell ist, möchten wir Ihnen keine Vorgaben machen. Erzählen Sie uns von Ihren Erfahrungen.	Misc Beim Einloggen nach Erstellen eines eigenen Accounts oben rechts Fehlermeldung "Could not load Favorites". Bedeutung CC, AA, AS, AE nicht intuitiv klar für Newbie. Show QAs- -> Show QA Templates Show Customer->Show Customers (Menü links) Update Customer bringt Fehler "Customer name already exists" (update passiert aber trotzdem) Dann "Error at creating customer." (wird aber trotzdem angelegt?) extendable -> extensible Default value - dropdown does not seem to work (bei Numeric Variable, mit range) Alle oder keine Category selektieren bringt "Could not

		create QAT" Update auf Template mit Variable bringt Fehlermeldungen Add QAT to Catalog does not seem to update "show cataloig" screen? ok after reload) QA ohne Variablen wird trotzdem zum "editieren" angeboten im Projekt Add QA on Project Level (after creation) does not seem to work Variablenreihenfolge wird beim Editieren der Werte vertauscht (???) Property-Anzeige "TRAMS" (Ladereihenfolge?) PDF und XML Export scheinen nicht zu funktionieren? JIRA Export schwer zu finden und scheint nicht zu funktionieren Help models -> model elements or key concepts Explanation -> Explanation "Software" -> "QUALI-T" (Namen verwenden) "This is the most important element in QUALI-T, because it is the original intention of the software. " -> hilfr mit nicht als User (und imho sind die QATs das Herz/der Kern der AW) +Bild (Domain MOdle einfach)? LIste/Tabelle unterteilen, erst QAT, Catalog, dann Project und QAI erklären? Textmenge ok, (W) und Content-Auswahl brauchen eine Iteration

		Change requests - eigene QA Categories - Name für QATs? "QA-ID" als Default - "Select a value" text configurable/editable? (Numeric Dropdown) - Fehlermeldungen konstruktiv machen - was soll User tun? Was ist root cause? . Ansicht der fertig editierten QAs (im Projekt)
19	Wir bitten Sie die Applikation vor Ende des UI-Reviews noch einmal durchzutesten. Da dieser Schritt nun generell ist, möchten wir Ihnen keine Vorgaben machen. Erzählen Sie uns von Ihren Erfahrungen.	[gleiche Frage wie zuvor?]
20	Wie gefällt Ihnen das Design der Applikation?	7
21	Wie intuitiv ist die Applikation?	7
22	Würden Sie die Applikation in der Zukunft brauchen wollen?	Ja
23	Für Rückfragen würden wir uns über Ihre E-Mail Adresse freuen	ozimmerm@hsr.ch

Tabelle 22 Antworten Usability-Test mit BA Betreuer

Einige Antworten (z.B. Frage 18) nicht ausgeführt werden konnte, haben wir mit dem BA-Betreuer die Situation in der letzten Woche der Bachelorarbeit noch einmal durchgespielt. Der Gesamteindruck hat sich gegenüber dem ersten Test stark verbessert.

3.4.3.2 Findings Usability-Test

Der Usability-Test wurde zeitgleich auch mit anderen Probanden durchgeführt. Das Resultat sind die nachfolgenden Findings. Diese konnten wir nicht mehr alle in die Applikation einbauen. Die Bugs haben wir jedoch behoben und alle weiteren Inputs könnten in eine Weiterentwicklung von QUALI-T einfliessen:

Finding	Kategorie	Status	Kommentar
Fehler beim Editieren eines ProjectInitiators: User existiert bereits	Bug	behoben	
Pop-ups rechts oben verschwinden nicht mehr und können nicht geschlossen werden	Bug	behoben	
Löschen von QAs funktioniert nicht	Bug	behoben	
Refresh nach Lösch- und Update-Operationen nicht immer automatisch	UI	behoben	

Change Role funktioniert nicht	Funktion	behoben	Rollenbasierte Menüs sind jetzt implementiert
Text in Suchfeld ist nicht informativ, zeigt nicht an, nach was allem gesucht werden kann	UI	behoben	Alle Objekte, nach denen gesucht werden kann, sind jetzt im Search-Textfeld erwähnt
Nutzen der Applikation für jemanden ohne Einführung und Hintergrundwissen nicht ganz klar	UI	behoben	Help in QUALI-T implementiert
Plus-Symbol beim Editieren eines Objekts ist verwirrend	UI	offen	Wird aber im Screencast, sowie in der Benutzeranleitung erklärt
IDs haben Teilweise ein -	UI	offen	Erklärung siehe Kapitel 3.3.3.3.2
Import von Bild beim Erstellen eines Catalog funktioniert nicht	Funktion	offen	
Export Project als PDF funktioniert nicht	Bug	offen	Dieser Bug tritt nur auf Heroku auf
Funktion Export Catalog als JSON ist nicht klar	UI	teilweise behoben	Die Funktion wird in der Dokumentation erläutert

Tabelle 23 Findings Usability-Test

4 Verzeichnisse

4.1 Tabellenverzeichnis

4.2 Abbildungsverzeichnis

4.3 Codeverzeichnis

4.4 Designentscheidungsverzeichnis

4.5 Literaturverzeichnis

[1] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran und P. America, «Generalizing a Model of Software Architecture Design,» in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, Pittsburgh, 2005.

[2] M. K. P. C. Rick Kazman, «ATAM: Method for Architecture Evaluation,» Carnegie Mellon University / Software Engineering Institute, Pittsburgh, 2000.

[3] M. Barbacci, M. H. Klein, T. A. Longstaff und C. B. Weinstock, «Quality Attributes,» Carnegie Mellon University / Software Engineering Institute, Pittsburgh, 1995.

[4] G. T. Doran, «There's a S.M.A.R.T. way to write management's goals and objectives,» *Management Review (AMA FORUM),* Bd. 70, Nr. 11, pp. 35-36, 1981.

[5] M. Morrison, «History of SMART Objectives,» RapidBI, 22 06 2010. [Online]. Available: https://rapidbi.com/history-of-smart-objectives/. [Zugriff am 15 06 2015].

[6] «ISO 25000 Software Product Quality,» [Online]. Available: http://iso25000.com/index.php/en/iso-25000-standards/iso-25010. [Zugriff am 10 06 2015].

[7] R. Fielding und J. Reschke, «RFC7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content,» Internet Engineering Task Force (IETF), 2014.

[8] A. Philipp, «AngularJS vs ReactJS for large web applications,» Liip AG, 16 09 2014. [Online]. Available: https://blog.liip.ch/archive/2014/09/16/angularjs-vs-reactjs-for-large-web-applications.html. [Zugriff am 11 06 2014].

[9] Dwayne, «When To Use AngularJS And When You Should Use ReactJS,» 28 10 2014. [Online]. Available: http://ilikekillnerds.com/2014/10/use-angularjs-use-reactjs/. [Zugriff am 11 06 2015].

[10] N. Thierry, «Faster AngularJS Rendering (AngularJS and ReactJS),» 19 04 2014. [Online]. Available: http://www.williambrownstreet.net/blog/2014/04/faster-angularjs-rendering-angularjs-and-reactjs/. [Zugriff am 11 06 2015].

[11] O. Tezer, «SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems,» DigitalOcean, 21 02 2014. [Online]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Zugriff am 10 06 2015].

[12] M. Widmer, «Trello Praxis-Check: Wohlwollende, aber kritische Gedanken mit Blick auf den Alltag,» förderland, 10 04 2013. [Online]. Available: http://www.foerderland.de/organisieren/news/artikel/trello-praxis-check-wohlwollende-aber-kritische-gedanken-mit-blick-auf-den-alltag/. [Zugriff am 10 06 2015].

[13] Wikipedia, «Pop-up notification,» Wikipedia, 14 05 2015. [Online]. Available: http://en.wikipedia.org/wiki/Pop-up_notification. [Zugriff am 10 06 2015].

[14] The JBoss Visual Design Team, «Hibernate Reference Documentation / Chapter 26. Best Practices,» Red Hat, 22 05 2013. [Online]. Available: https://docs.jboss.org/hibernate/core/4.3/manual/en-US/html/ch26.html. [Zugriff am 01 06 2015].

[15] B. Polasek, M. Cicolini und AdNovum Informatik AG, «Essentials of Testing,» Jazoon, 01 06 2010. [Online]. Available: http://jazoon.com/history/Portals/0/Content/slides/th_a6_1030-1120_polasek.pdf. [Zugriff am 10 06 2015].

[16] B. Nadel, «HTTP Status Codes For Invalid Data: 400 vs. 422,» 17 10 2012. [Online]. Available: http://www.bennadel.com/blog/2434-http-status-codes-for-invalid-data-400-vs-422.htm. [Zugriff am 10 06 2015].

[17] «Top Ten Lists of Software Project Risks,» 16-18 März 2011. [Online]. Available: http://www.uio.no/studier/emner/matnat/ifi/INF5181/h14/pensumliste/microsoft-word---

iaeng-top-ten-lists-of-software-project-risk1---imecs2011_pp732-737.pdf. [Zugriff am 22 09 2014].

[18] O. Zimmermann, «Application Architecture – Einführung und Vorlesungsüberblick,» *Vorlesung Application Architecture HS2014,* pp. 33-35, 18 09 2014.

4.6 Glossar

Abkürzung / Begriff	Beschreibung
API	Application Programming Interface, Programmierschnittstelle
BA	Bachelorarbeit
BSD	Berkeley Software Distribution
CI	Continuous Integration, Software-Engineering-Praxis, um eine durchgehend getestete Software zu erreichen
CLI	Command Line Interface
CRUD(S)	Create, Read, Update, Delete (Search)
ECTS	European Credit Transfer System
FA / FR	Funktionale Anforderung / Functional Requirements
FOB	Formatting Objects Proccesor
FQN	Fully Qualified Name
GPL	General Public License
LGPL	Lesser General Public License
HTML	Hypertext Markup Language
HTTP(S)	Hypertext Transfer Protocol (Secure)
Issue	Ein Ticket im Issue-Management-Tool JIRA
JIRA	Projekt-Management-/Issue-Tracking-Tool (wird während der BA zur Zeiterfassung und Projektplanung verwendet)
JS	JavaScript, Client-seitige Skriptsprache
JSON	JavaScript Object Notation
MIT	Massachusetts Institute of Technology
NFA / NFR	Nichtfunktionale Anforderung / Non Functional Requirements
QA	Quality Attribute, Synonym für NFR bzw. NFR
RFC	Request for Comments
SA	Studienarbeit
SBT	Simple Build Tool (analog zu Javas Ant) für Scala
SMART	Qualitätseigenschaften für NFRs
SQL	Structured Query Language
XSL-FO	Extensible Stylesheet Language – Formatting Objects
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Tabelle 24 Glossar mit Abkürzungen und Beschreibungen