Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Darjan Meyerhoff

Konzeptionelles Design und Realisierung eines Multi-Plattform Systems zur Bereitstellung und Verwaltung von Betriebsmitteln in einer Unternehmensumgebung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Darjan Meyerhoff

Konzeptionelles Design und Realisierung eines Multi-Plattform Systems zur Bereitstellung und Verwaltung von Betriebsmitteln in einer Unternehmensumgebung

Bachelorarbeit eingereicht im Rahmen Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Olaf Zukunft

Abgegeben am 06.05.2016

Darjan Meyerhoff

Thema der Arbeit

Konzeptionelles Design und Realisierung eines Multi-Plattform Systems zur Bereitstellung und Verwaltung von Betriebsmitteln in einer Unternehmens-umgebung

Stichworte

Informationssystem, Datenmanagement, verteiltes System, Multi-Plattform, Ruby on Rails, Android

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Konzeptionierung und der Realisierung eines verteilten Informationssystems, welches sowohl die administrative Verwaltung von Betriebsmitteln als auch für die teamübergreifende Verteilung dieser Ressourcen innerhalb eines Unternehmens unterstützt.

Darjan Meyerhoff

Title of the paper

Conceptual design and realization of a multi-platform system for provisioning and management of equipment in a corporate environment

Keywords

information system, data management, distributed computing, multi-platform, Ruby on Rails, Android

Abstract

This paper addresses the conception design and realization of a distributed information system, which supports both the administrative management of resources as well as for cross-team distribution of those resources within a corporate environment.

Inhaltsverzeichnis

1 Einleitung

1.1 Zielstellung

Ziel der Arbeit ist es ist die Entwicklung eines verteilten Informationssystems als maßgeschneiderte Lösung für die administrative Verwaltung von testbezogenen Betriebsmitteln und die teamübergreifende Verteilung dieser Betriebsmittel innerhalb einer Unternehmensumgebung. Mit diesem System sollen die Arbeitsabläufe innerhalb des Unternehmens bei der Verwaltung und der Verteilung von Betriebsmitteln vereinfacht und verbessert.

1.2 Gliederung der Arbeit

Mit der Beschreibung der einzelnen Kapitel soll ein Überblick über den Aufbau der Arbeit gegeben werden.

Kapitel 1 Das einleitende Kapitel beschreibt die Ziele der Arbeit und gibt einen Überblick auf die Struktur der Arbeit.

Kapitel 2 Dieser Abschnitt der Arbeit befasst sich mit den Anforderungen, die an das entwickelte System gestellt werden.

Kapitel 3 In diesem Kapitel wird auf grundlegende Werkzeuge, Konzepte und Methoden erörtert, welche für ein Verständnis der Arbeit von Bedeutung sind.

Kapitel 4 Innerhalb dieses Kapitels wird auf die Konzeptionierung des Systems eingegangen und anhand wichtiger Komponenten des Systems gezeigt.

Kapitel 5 In diesem Teil der Arbeit werden interessante Teile der Realisierung und Optimierung des Systems vorgestellt.

Kapitel 6 Zusammenfassend zeigt dieses Kapitel die wichtigsten erreichten Ziele und geht mögliche Zukunftsperspektiven des Systems ein.

2 Anforderungsanalyse

Dieses Kapitel behandelt, wie die Arbeitsabläufe zur Verwaltung der Betriebsmittel im Unternehmen zu Beginn der Arbeit umgesetzt worden sind und geht ein auf die Anforderungen an das System, welches im Zuge dieser Arbeit entwickelt werden soll.

2.1 Ist - Zustand

Das System zur Verwaltung von geliehenen Testgeräten, welches bisher im Unternehmen genutzt wird, ist im Zuge einer sogenannten Hackweek durch zwei Mitarbeiter entwickelt worden. In einer solchen Hackweek, darf sich jeder Mitarbeiter eine Woche lang einem frei gewählten Projekt widmen und dieses so weit wie möglich umsetzen.
Das bestehende System bietet eine Liste aller Testgeräte mit einigen grundlegenden Informationen zu jedem einzelnen Testgerät. So wird zu einem Gerät erfasst, welches Betriebssystem installiert ist und ob es sich um ein Smartphone oder ein Tablet handelt. Darüber hinaus existiert zu jedem Gerät ein Feld für Anmerkungen. Auf Grund der der beschränkten Möglichkeit detaillierte Informationen zu einem Gerät an dafür vorgesehenen Stellen zu hinterlegen, kann diese Anmerkungszeile sehr unterschiedliche Informationen beinhaltet. Abhängig davon, welcher Mitarbeiter ein neues Gerät in das System einträgt, werden hier technischen Details, die Seriennummer, das Team welches das Gerät lagert, oder aber die Farbe eines Gerätes festgehalten. Zum Teil wird in dieses Feld gar keine Information eingetragen. Da das System keine übersichtlichen und aus Platzmangel in der Darstellung auch keine detaillierten Informationen erlaubt, wird parallel mit einer Excel-Tabelle gearbeitet, in welcher diese Informationen festgehalten werden. Diese ist nur den Administratoren des Systems zugänglich. Zusätzlich zu den Geräteinformationen kann ein Gerät als geliehen markiert werden, wobei ein voraussichtlicher Rückgabetermin und die Person hinterlegt ist, welche das Gerät ausgeliehen hat.
Die Betriebsmittel werden in einem abschließbaren Schrank gelagert, aus welchem die Mitarbeiter unter Absprache mit den Administratoren des bestehenden Systems Geräte auswählen können, die sie gerne Leihen möchten. Diese Geräte werden dann als geliehen markiert und mit Namen des leihenden Mitarbeiters und einem

Rückgabetermin versehen. Die Rückgabe der Geräte erfolgt analog dazu durch Löschung der Markierung. Für Administratoren ist es nicht möglich Informationen über vergangene Leihgaben einzusehen.

2.2 Soll - Zustand

Das neue System soll das Bisherige komplett ablösen und in sich geschlossen alle nötigen Funktionen abbilden, die sich im Laufe der Zeit bei der Verwaltung und Verteilung der Betriebsmittel zum Testen im mobilen Entwicklungsbereich als Notwendig erwiesen haben. Die durch das System angebotenen Betriebsmittel sollen nicht wie bisher nur die Testgeräte umfassen, sondern um zum Beispiel Zubehör, Sim Karten und anderen Betriebsmitteln erweitert werden, die von den Teams im Mobile Bereich des Unternehmens zum Testen benötigt werden.
Im Zuge der Planung von Testsessions, zum Teil mit unternehmensexternen Testern, welche schon eine gewisse Zeit im Voraus geplant werden müssen und auch sehr kostenintensiv sein können, haben sich weitere Anforderungen an das neue System ergeben. Um Mitarbeiter bei dieser Planung zu unterstützend, soll es ihnen einerseits mit Hilfe des neuen Systems ersichtlich sein, ob die für die Testsession benötigten Betriebsmittel zu dem von ihnen gewünschten Zeitraum zur Verfügung stehen. Andererseits soll es ihnen nun auch möglich sein diese Betriebsmittel für einen angegebenen Zeitraum im Vorfeld zu reservieren.
Zum einen soll das System eine Schnittstelle in Form einer Weboberfläche anbieten, die von jedem Rechner überall im Unternehmen einfach und flexibel über einen beliebigen Webbrowser erreicht werden kann. Über die Weboberfläche soll es jedem Benutzer möglich sein einen Überblick über die zur Verfügung stehenden Betriebsmittel zu erhalten. Zu den einzelnen Betriebsmitteln sollen abhängig von ihrer Kategorie alle Informationen ersichtlich sein, die für die Nutzer der unterschiedlichen Teams relevant sind. Zum Beispiel ist es für einen Entwickler der mobilen Plattform des Unternehmens wichtig, schnell und einfach ein Testgerät auswählen zu können, welches bestimmte Kriterien erfüllt, wie Typ und Hersteller des Geräts, installiertes Betriebssystem und die Version des Betriebssystems. Um dies zu ermöglichen muss das System Sortierungs- und Filteroptionen zu den Informationen der Betriebsmittel anbieten. Hier ist hervorzuheben, dass sich das Spektrum an Betriebsmitteln im Laufe der Zeit ändern und auch erweitern kann. Aus diesem Grund muss bei der Entwicklung des Systems darauf geachtet werden, dass es einfach ist weitere Kategorien von Betriebsmitteln zu erstellen und bestehende Kategorien abzuändern.

Benutzer mit administrativen Rechten zu dem System sollen in der Lage sein, die Betriebsmittel über die Weboberfläche zu verwalten. Hierfür können sie Betriebsmittel in das System aufnehmen und die Informationen von bereits registrierte Betriebsmittel abändern. Das System muss durch die Validierung von Pflichtfeldern sicherstellen, dass zwingend Notwendige Informationen bei der Erstellung und Änderung von Betriebsmitteln angegeben werden. Für Administratoren soll es möglich sein, sich einen schnellen Überblick über die geliehenen und reservierten Betriebsmittel zu verschaffen. Aus an dieser Stelle soll der Überblick durch Filterung und Sortierung der Filterergebnisse so einfach wie möglich zum gewünschten Ziel führen. Da die Administratoren des alten Systems des Öfteren von Mitarbeitern um Beratung bei der Auswahl von Testgeräten gebeten werden, soll es diesen Administratoren mit dem neuen System möglich sein, bei Bedarf im Namen anderer Mitarbeiter Betriebsmittel reservieren und leihen zu können. Damit sich Administratoren einen Überblick verschaffen können, soll das System Protokolle und Statistiken zu unterschiedlicher Aktionen innerhalb des Systems führen und den Administratoren über die Weboberfläche leicht zugänglich machen. Beispiele hierfür sind die Anzeige aller jemals getätigten Ausleihaktionen und eine Statistik darüber, wie zuverlässig ein User sich an die vereinbarten Rückgabetermine gehalten hat, da bei dem bisherigen System auf Grund der fehlenden Übersichtlichkeit durchaus mal vorkam, dass alle Beteiligten die Rückgabe vergessen haben. Auch für einzelne Betriebsmittel soll das neue System aufzeigen, in welcher Häufigkeit diese Benutzt werden, um es den Administratoren zu ermöglichen, dementsprechend oft genutzte Betriebsmittel in größerer Anzahl anzubieten.
Für nicht administrative Benutzer soll es an der Weboberfläche möglich sein die von ihnen reservierten und geliehenen Betriebsmittel einzusehen. Es soll ihnen auch möglich sein, Reservierungen für Betriebsmittel zu erstellen und zu stornieren.
Um die Interaktion der Mitarbeiter mit dem System am Lagerungsort der Betriebsmittel so einfach wie möglich zu gestalten, wurde innerhalb des ersten Spezifikationsmeetings beschlossen, am Lagerort der Betriebsmittel einen Zugangspunkt zum System anzubieten. Über diesen Zugangspunkt soll es Benutzern des Systems möglich sein, eine Auswahl von Betriebsmitteln für den Verleih, die Reservierung oder für die Rückgabe am System zu registrieren und von dem System Feedback zu dem aktuellen Vorgang zu erhalten. Der Zugangspunkt soll technisch mit einem mobilen Endgerät realisiert werden, welches am Lagerort angebracht wird. Damit dies möglich ist, muss das System als ein verteiltes System konzipiert werden um Teile der Funktionalität des Systems über das mobile Endgerät anbieten zu können. Da innerhalb des Unternehmens nach und nach die Situation entsteht, das Betriebsmittel nicht mehr zentral an einem Ort gelagert werden, einzelne Teams für

sie im täglichen Geschäft relevante Betriebsmittel gerne selber lagern und verwalten wollen, sollte es möglich sein mehrere solcher mobilen Endgeräte parallel betreiben zu können, um alle Lagerorte abzudecken.
Um die Benutzer über sie relevante Ereignisse innerhalb des Systems zu informieren, soll eine Komponente zur Verteilung von E-Mails in das System integriert werden. Diese soll die Benutzer zum Beispiel daran erinnern, dass sie Betriebsmittel rechtzeitig zurück bringen müssen oder reservierte Betriebsmittel auf die Abholung warten. Darüber hinaus versendet diese Komponente Bestätigungen, zum Beispiel über getätigte Reservierungen und Leihaktionen.

2.3 Wahl der Entwicklungsumgebung

Das System wird nach Abschluss der Arbeit an das Unternehmen übergeben und wird ab diesem Zeitpunkt von den Mitarbeitern gewartet und möglicherweise auch in seinem Funktionsumfang erweitert. Damit dies reibungslos möglich ist wurde in dem ersten Spezifikationsmeeting beschlossen, das System mit Hilfe einer Entwicklungsumgebung zu realisieren, welche von den Entwicklern des Unternehmens im täglichen Gebrauch ist.
Da die Web-Plattform des Unternehmens und deren Anwendungskern mit Ruby on Rails[ROR1] realisiert wurde, bietet es sich dementsprechend an, auch den Anwendungskern und die Weboberfläche des neuen Systems mit Hilfe von Rails umzusetzen.
Sowohl das System als auch die Betriebsmittel werden innerhalb des Unternehmens wie bisher hauptsächlich von den Entwicklern des Android Team betreut und verwaltet. Aus diesem Grund wurde entschieden die Komponente zur Registrierung von Betriebsmittel am Lagerort auf einem Android Gerät in Form einer Android Applikation zu realisieren.

3 Grundlagen

Dieses Kapitel behandelt Konzepte und Methoden, die im Rahmen der Arbeit für die Konzipierung und Realisierung des Systems relevant sind. In diesem Zusammenhang wird eine Auswahl von möglichen Entwicklungsmodellen vorgestellt und auf die Grundlagen für die Entwicklung einerseits mit Ruby on Rails und anderseits mit Android eingegangen.

3.1 Testgetriebene Entwicklung

Bei den klassischen Entwicklungsmodellen, so zum Beispiel beim V-Modell, wird das Gesamtsystem beim Entwurf in immer kleinere Komponenten aufgeteilt. Diese Komponenten werden Implementier und auf die Erfüllung der Anforderungen die sich aus der Spezifikation für diese Komponenten ergeben getestet.
Bei der testgetriebenen Entwicklung erfolgt die selbe Aufteilung des Gesamtsystems, aber im Gegensatz zu den klassischen Entwicklungsmodellen wird hier die Testfallerstellung auf Basis der Anforderungen der Spezifikation der eigentlichen Implementierung der Komponenten vorangestellt. [SPIL10]
Der Vorteil der testgetriebenen Entwicklung besteht darin, dass bereits während der Erstellung der Testfälle anhand der Spezifikation bereits im Vorfeld überlegt werden muss, was genau durch die Implementierung umgesetzt soll und was nicht. Hierdurch wird sichergestellt, dass der Code besonders zielgerichtet zur Umsetzung der Anforderungen der Spezifikation führt und nicht durch nicht zielführende Bestandteile überladen wird. Darüber hinaus ist durch die kleinschrittige Erweiterung um Tests und dazugehörigen Implementierungen eine hohe Testabdeckung gewährleistet.
Um diese Iteration aus Testfallerstellung, Implementierung und Evaluierung zu vereinfachen und bei einer Änderung am Code sofort sehen zu können, ob die Testbasis und damit das erwartete Verhalten weiterhin erfüllt bleibt, bietet es sich an die Testausführung zu automatisieren.

3.1.1 Verhaltensgetriebene Entwicklung

Die verhaltensgetriebene Entwicklung kann als Weiterführung der testgetriebenen Entwicklung angesehen werden. Auch bei ihr gilt das Grundprinzip, erst die Testfälle zu erstellen und dann die Komponenten zu implementieren. Im Gegensatz zur testgetriebenen Entwicklung basieren die Testfälle ausschließlich auf dem in der Spezifikation festgelegten Verhalten des Systems und seiner Komponenten. Das bedeutet die Grundlegende Frage bei der Umsetzung einer Anforderung ist nicht wie sie umgesetzt werden soll, sondern was für ein Verhalten vom System erwartet wird. Die Testfälle werden also komplett unabhängig von der Struktur und den technischen Details des zu testenden Codes erstellt. [CHEL10]

Durch die Entkopplung von Testfällen und Implementierungsdetails ist sichergestellt, dass bei Veränderungen am System im Laufe der Entwicklung, und hier im speziellen bei der Veränderung von technischen Implementierungsdetails, stets das erwartete Verhalten weiterhin bestehen bleibt. Idealerweise sollte eine solche Änderung nicht dazu führen, dass die Tests angepasst werden müssen. Der Fokus auf das Testen des Verhaltens verbessert sowohl die Wartbarkeit als auch die Erweiterbarkeit eines Systems.

Bei der Benennung der Testfälle ist bei der verhaltensgetriebenen Entwicklung auf eine natürlich sprachliche Formulierung zu achten, die das zu testenden Verhalten beschreibt. Im Falle eines fehlgeschlagenen Tests ist dann sofort ersichtlich, welches erwartete Verhalten von einer Komponente nicht erfüllt wurde.

Um insgesamt eine gute Testabdeckung des Codes zu erhalten, bietet es sich an Testfälle der test- und der verhaltensgetriebenen Entwicklung zu kombinieren.

3.1.2 Einordnung innerhalb der Entwicklungsmodelle

Wie bereits angesprochen ist in klassischen Entwicklungsmodellen wie dem Wasserfall-Modell und dem V-Modell der Softwaretest stets der letzte Schritt zur Evaluierung einer Entwicklungsphase. Dies steht zwar im kompletten Kontrast zu dem Prinzip der test- und verhaltensgetriebenen Entwicklung, es ist aber prinzipiell möglich durch Umstellung der Phasen dieser Entwicklungsmodelle dem Paradigma der test- beziehungsweise verhaltensgetriebenen Entwicklung zu entsprechen. Da beim V-Modell bei der Konzeptionierung eines Systems die Anforderungen und Komponenten beim durchlaufen der Entwicklungsphasen immer feiner granuliert werden, eignet sich dieses Entwicklungsmodell sehr gut für die Anpassung an die iterative test- beziehungsweise verhaltensgetriebene Entwicklung.

Im Bereich der modernen agilen Entwicklungsmodelle, wie Kanban und Scrum, liegt der Fokus darin, Entwicklungseinheiten so fein wie möglich zu gliedern unter

anderem, um Prozesse der Entwicklung zu parallelisieren und zu vereinfachen. In diesem Bereich ist sowohl test- als auch verhaltensgetriebene Entwicklung bereits ein fester Bestandteil der Softwareentwicklung.

3.2 Entwicklung mit Ruby on Rails

Wie in der Anforderungsanalyse deutlich wird, soll ein Teil des Systems mit Hilfe von Ruby on Rails entwickelt werden. In diesem Abschnitt werden wichtige Methoden und Komponenten vorgestellt, welche bei der Entwicklung dieses Teilsystems System relevant sind.

Rails ist ein Framework zur Entwicklung von Web Applikationen. Es ist nach dem Model Viewer Controller Pattern, kurz MVC, aufgebaut. Hierbei repräsentiert das ActiveRecord die Umsetzung des Models, der ActionController die Umsetzung des Controllers und der ActionView und seine Komponenten Template, Partial und Layout die Umsetzung der View.

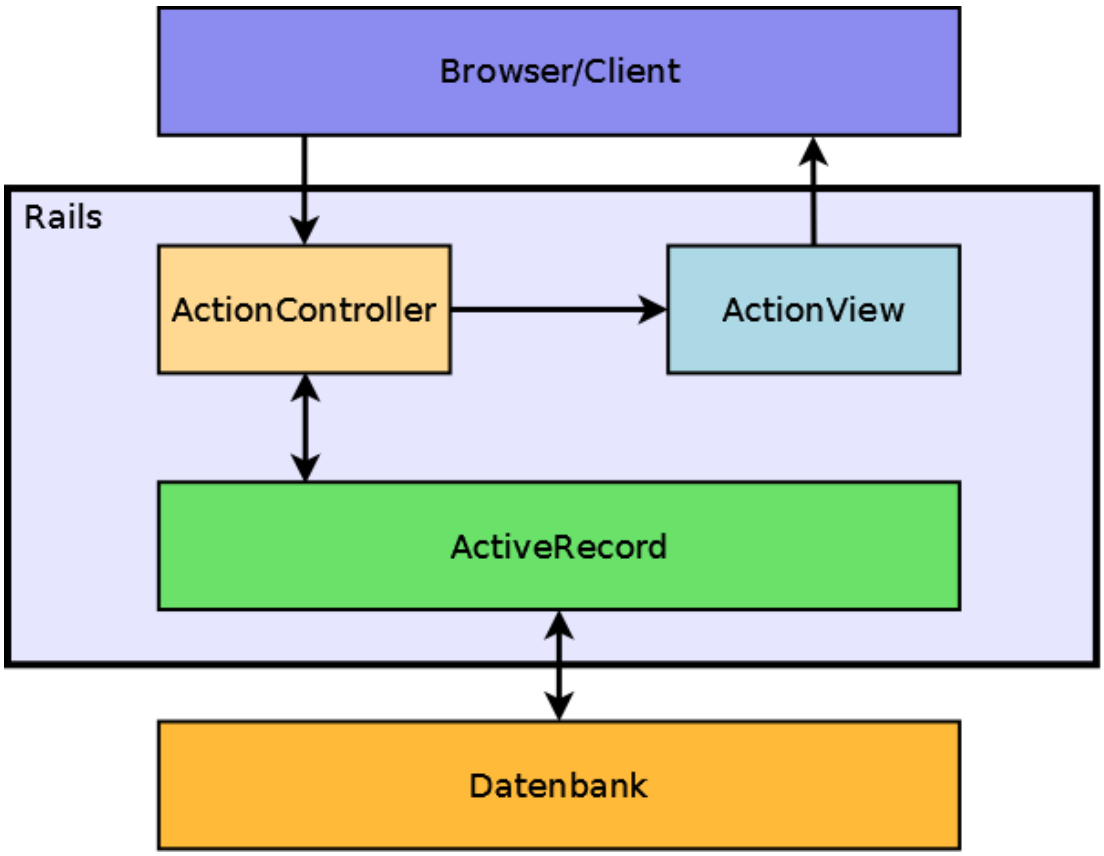

Abbildung 1 - Vereinfachte Systemarchitektur von Ruby on Rails

Die Abbildung 1 - Vereinfachte Systemarchitektur von Ruby on Rails zeigt einen groben Überblick über die innere Struktur einer Rails Anwendung und beschreibt den Workflow vom Aufruf durch einen Client über die Verarbeitung durch die Komponenten bis hin zur Antwort. Für einen detaillierteren Einblick wird in den folgenden Abschnitten auf die Rollen und Funktionsweisen der einzelnen Komponenten innerhalb einer Rails Anwendung genauer eingegangen.

3.2.1 ActiveRecord

Das ActiveRecord [ROR2] ist eine objektrelationale Abbildung der Modelle einer Rails Anwendung auf die verwendete Datenbank. Anhand eines Schemas wird angegeben, welche Attribute eines Models auf die Datenbank abgebildet werden sollen und mit welchem Datentypen diese Attribute versehen werden sollen. Die Relation zwischen verschiedenen Modellen lässt sich innerhalb des ActiveRecord unabhängig von der verwendeten Datenbank definieren und wird durch Rails auf die Datenbank abgebildet. Um die Abbildung von Instanzen von ActiveRecords auf Datensätze eines konkreten Datenbanksystems zu ermöglichen, besitzt Rails eine Reihe von Adaptern. Aus Sicht der Anwendung wird hiermit komplett von der zu Grunde liegenden Datenbank abstrahiert.
Neben den Grundlegenden Datenbankfunktionen wie CRUD und Abfragen an die Datenbank, bietet ActiveRecord weitere Funktionen an, um die Umsetzung der Modelle zu erleichtern. Für die Attribute eines Models bietet ActiveRecord einen Validierungsmechanismus. Durch die Validierung stellt Rails sicher, dass eine Instanz nur dann persistent gespeichert wird, wenn durch die Werte der Attribute definierte Invarianten nicht verletzt werden. Dies kann eine einfache Längenprüfung eines String-Attributes sein. Die Prüfung kann aber auch beliebig komplex sein und die Attribute eines Models in Abhängigkeit von Relationen des Models überprüfen. Darüber hinaus haben ActiveRecords einen sogenannten Lebenszyklus. Dieser Lebenszyklus beschreibt den Verlauf, den eine Instanz eines ActiveRecords von der Erstellung über seine Speicherung und Änderung bis hin zur Löschung beschreibt. Mit Hilfe von Callback Methoden ist es möglich zu bestimmten Abschnitten innerhalb des Lebenszyklus vordefinierte Funktionen aufzurufen. Dies ermöglicht zum Beispiel die Bereinigung der Datenbank, indem bei der Löschung eines Datensatzes auch all diejenigen Datensätze gelöscht werden, welche nur in Verbindung mit dem ersten Datensatz Relevanz haben. Ein Beispiel wäre eine Adresse welche nur in Verbindung mit der dazugehören Benutzer von Bedeutung ist. Denkbar wäre auch der

gegenläufige Fall, in dem bei der Erstellung eines Benutzers automatisch die abhängige Adresse erstellt werden würde.
Durch das Ableiten von ActiveRecord besitzen die Modell, welche innerhalb eine Rails Anwendung implementiert werden, bereits ein breites Spektrum an Basisfunktionalität, schon bevor diese bei der Entwicklung mit modelspezifischen Funktionen erweitert werden. So ist es möglich, mit Hilfe von ActiveRecords die Verwandtschaft von Modellen, im Sinne einer Vererbungshierarchie, abzubilden. Das Verhalten eines Models, welches auf ActiveRecord basiert, lässt sich also wiederum durch Vererbung an ableitende Klassen weiterreichen. Am Beispiel der in der Anforderungsanalyse angeführten Modelle Device, Sim Card und Accessory, wäre es möglich bei der Implementierung der Modelle ihr gemeinsames Verhalten in einer abstrakten übergeordneten Basisklasse Item zu bündeln. Im Hinblick auf die Vererbung von Methoden und Strukturen innerhalb der Klasse müssen bei dem ActiveRecord, im Vergleich zu anderen Ruby Klassen, keine zusätzlichen Maßnahmen getroffen werden. Interessant ist aber die Frage, wie die Vererbungshierarchie von ActiveRecords nun auf Ebene der Persistenz und damit innerhalb des untergeordneten Datenbankmodells implementiert werden muss. Für die Abbildung auf die Datenbank gibt zwei grundlegend unterschiedliche Konzepte: Single Table Inheritance, kurz STI und Multi Table Inheritance, kurz MTI [ROR3]. Bei der Wahl des richtigen Konzeptes müssen die Vor- und Nachteile in Bezug auf das Projekt gegeneinander abgewogen werden.
Bei der STI werden alle Instanzen von ActiveRecords die zu einer Vererbungshierarchie gehören in einer Datenbanktabelle gespeichert. Hierbei beinhaltet die Tabelle, welche die Basisklasse der Hierarchie repräsentiert, alle Datensätze, sowohl der Basisklasse als auch aller ableitenden Klassen. Diese Tabelle wird ergänzt um ein Typ Attribut, welches zu jedem Datensatz die Information beinhaltet, um welche konkrete Klasse es sich bei diesem handelt. Damit dies möglich ist, muss die Tabelle der Basisklasse für jede ableitende Klasse stets um die Attribute dieser neuen Klasse erweitert werden. Die resultierende Tabelle besitzt also die Summe aller Attribute der Basisklasse und aller ableitenden Klassen. Hierdurch besitzt auch ein einzelner Datensatz, unabhängig davon welche Klasse er repräsentiert, aus Sicht der Datenbank sämtliche Attribute. Die Attribute, welche eine Klasse aus Sicht des Models nicht besitzt, werden auf Datenbankebene mit null-Werten versehen. Hier wird ein Problem des STI Konzeptes deutlich. Bei jeder Erweiterung der Vererbungshierarchie um weitere ableitende Klasse, wird die gesamte Tabelle und damit jeder einzelne bereits existierende Datensatz aller verwandten Klassen auch um diese Attribute erweitert, wodurch die Tabelle immer weiter in die breite „wächst“. Auch wenn in diesem Fall viele Felder innerhalb der

Tabelle mit null belegt wären, so kann dieses Verhalten bei einer großen Menge von Datensätzen einen negativen Einfluss auf die Skalierbarkeit der Anwendung nehmen. Dies ist ein klarer Nachteil de STI Konzeptes. Durch die Zusammenlegung aller Klassen einer Vererbungshierarchie in eine Tabelle, können Datenbankabfragen, die sich auf die gesamte Hierarchie beziehen sehr einfach und performant umsetzen lassen. Bezogen auf das Beispiel Item und die ableitenden Klassen Device, Sim Card und Accessory ist es bei einer Abfrage unerheblich ob sich die Abfrage auf eine Teilmenge von Klassen bezieht oder auf die Vererbungshierarchie Item im Ganzen, es bleibt eine Abfrage an eine Datenbanktabelle. Zusammengefasst bietet STI eine gute Leistung bei Datenbankabfragen zu Kosten der Skalierbarkeit der Datenbankgröße.
Demgegenüber steht das Konzept der MTI. Basierend auf dem Problem der Datenbankgröße, welche sich bei der Bündelung aller Klassen in einer Tabelle ergebt, besitzt bei MTI die Basisklasse und jede ableitende Klasse jeweils eine eigene Tabelle. Es werden also nur die Attribute auf die Datenbank abgebildet, die eine Klasse wirklich besitzt. Dieser Ansatz ist bei der Erweiterung der Vererbungshierarchie sehr viel skalierbarer als STI, denn für neue ableitende Klasse wird nur eine weitere Tabelle angelegt. Die Aufteilung der Vererbungshierarchie in mehrere Tabellen wirft aber auch Probleme auf. Für jede einzelne Datenbankabfrage, welche sich auf mehr als eine der Klassen der Vererbungshierarchie bezieht, muss die Datenbank die zu Grunde liegenden Tabellen mit einer FULL INNER JOIN Operation zusammenlegen. Diese Operation kostet abhängig von der Anzahl der involvierten Tabellen und Datensätze viel Rechenleistung. Zusammenfassend bietet MTI eine kompaktere Größe der Datenbank zu Kosten von Rechenleistung bei Operationen, die sich auf eine Teilmenge Vererbungshierarchie beziehen.
Bei der Wahl des Konzeptes zur Umsetzung einer Vererbungshierarchie mit ActiveRecords sollten die Vor- und Nachteile der beiden Konzepte in Abstimmung mit den erwarteten Datenbankoperationen genau abgewogen werden.

3.2.2 ActionController

Die Interaktion zwischen einem Benutzer beziehungsweise einem System und der Rails Anwendung erfolgt ausschließlich über das HTTP Protokoll in Form eines URL Aufrufs. Die URL setzt sich dabei aus der Adresse des Webservers und einem Pfad zusammen. Im Gegensatz zu klassischen HTML-basierten Webservern dient der Pfad nicht für die Anfrage eines statischen HTML Dokuments, sondern entspricht einer Interaktion mit der Rails Anwendung.
Für die Umsetzung der Anwendungslogik, und damit der Funktionalität, die eine Rails Anwendung nach Außen anbietet, ist der ActionController [ROR4] zuständig. Die

Möglichen Anfragen, die ein ActionController anbietet wird mit Hilfe von Methoden umgesetzt, welche Actions genannt werden. Bei einer Interaktion mit der Anwendung wertet der Dispatcher den Pfad des Aufrufs aus. Anhand einer Routing Tabelle [ROR5] kann der Dispatcher dieser URL entnehmen, welcher ActionController und auch welche Action innerhalb des ActionControllers aufgerufen werden soll. Bei einem Aufruf können zusätzlich zum Pfad auch Parameter innerhalb der URL angegeben werden. Der ActionController wird Instanziiert und die referenzierte Action wird zusammen mit möglichen Parametern ausgeführt. Diese Action setzt die von ihr implementierte Funktionalität um und generiert eine Antwort auf die Anfrage. Die Gestalt der Antwort ist in der Regel abhängig von dem Format, auch MIME-Type genannt, das durch den Aufruf angefragt wird. So kann ein und dieselbe Action Abhängig von der Anfrage wahlweise ein HTML Dokument erzeugen, oder aber auch einen JSON String.
Bei der Entwicklung einer Rails Anwendung sollten der Funktionsumfang, der von den einzelnen ActionControllern umgesetzt wird klar voneinander getrennt sein, nach dem Prinzip des single point of concern. So ist es zum Beispiel sinnvoll, alle Anfragen mit Bezug auf ein bestimmtes ActiveRecord innerhalb eines ActionControllers der Rails Anwendung zu bearbeiten.

Während der ActionController dafür zuständig ist, einen Teil Anwendungslogik umzusetzen und die Daten, welche für die Antwort benötigt werden vorzubereiten, delegiert er innerhalb des Render-Prozesses an die ActionView.

3.2.3 ActionView

ActionViews [ROR6] sind die Basis für die Generierung des Inhaltes einer Antwort. Eine ActionView kann hierbei sowohl statische als auch dynamische Komponenten beinhalten. Der dynamische Teil einer ActionView wird mit Hilfe von Ruby Code umgesetzt. Damit der Render-Prozess statischen Inhalt von dynamischen Ruby Code unterscheiden kann, wird der Ruby Code mit einer einleitenden und einer beendenden Symbolfolge markiert. Dieser eingebettete Ruby Code wird auch Embedded Ruby genannt. Der Render-Prozess führt den Ruby Code im Programm Kontext des ActionControllers aus, von welchem der Render-Prozess aufgerufen wurde und hat dabei Zugriff auf alle lokalen, Instanz weiten und globalen Variablen des aufrufenden ActionControllers. Das Ergebnis der Auswertung ersetzt innerhalb des ActionViews nun die Stelle des Embedded Ruby. Im Anschluss an die Auswertung und Ersetzung aller dynamischen Komponenten liegt der Inhalt der Antwort vor.

Es wird zwischen drei Arten von ActionViews unterschieden, Layouts, Templates und Partials. Syntaktisch gibt es kaum einen Unterschied zwischen diesen Arten von ActionViews, denn sie alle erlauben die Einbindung von sowohl statischen als auch dynamischem Elementen. Semantisch erfüllen sie jeweils eine andere Rolle innerhalb des Render-Prozesses.
Layouts stehen über den Templates und beinhalten Informationen, welche eine Art Grundlegenden Rahmen für mehrere, wenn nicht alle ActionController übergreifenden Antworten beinhaltet. In das Layout wird ein Template eingebunden. In aller Regel besitzt eine Action eines ActionControllers ein eigenes Template, welches als Antwort auf einen Aufruf der Action gerendert wird und alle Informationen besitzt, die durch die Action bereitgestellt werden sollen.
Partials sind für die Darstellung eines Teilbereichs von Informationen zuständig und können beliebig in Template und auch andere Partiels eingebunden werden.
Die Komposition aus Layout, Template und Partials erlaubt jede beliebige Darstellung von Informationen unter Beachtung des don't repeat yourself Programmierparadigmas.
Am Beispiel einer Webshop Onlinedienstes, könnte das Layout alle Elemente beinhalten, welche auf jeder Seite angezeigt werden, wie zum Beispiel Navigationsleisten und das Logo des Webshops. Bei dem Aufruf einer Übersicht zu einem Benutzer dieses Shops würde innerhalb dieses Layouts ein Template mit allen Informationen zu diesem User gerendert werden. Dieses Template könnte wiederum ein Partial beinhalten, welches die Informationen zu der Adresse dieses Users beinhaltet.

3.2.4 Die Rolle von Rails innerhalb des Systems

Betrachtet man heutige Webdienste, so sieht man, dass viele Anbieter parallel zu einer für Benutzer mit Hilfe eines Browsers konsumierbaren HTML-basierten Schnittstelle, eine weitere für Maschinen verwertbare API Schnittstelle anbieten. Als Beispiele sind hier die Webdienste von Google [GOOA], GitHub [GITA], Facebook [FACA] und Twitter [TWIA] zu nennen, welche neben ihren Internetseiten zum Teil öffentliche, zum Teil aber auch kostenpflichtige API Schnittstellen anbieten.
Der Anwendungskern, der innerhalb des Systems implementiert wird, bietet zwei Schnittstellen nach außen an. Auf der einen Seite eine Weboberfläche für die Interaktion mit den Benutzern und zum anderen eine API Schnittstelle für die Interaktion mit der auf Android basierenden Registrierungskomponente.
Für die Realisierung einer API Schnittstelle bietet Rails eine Reihe von Mechanismen an, welche die Entwicklung einer solchen API Schnittstelle unterstützen. Mit Hilfe der

Routingtabelle ist es möglich, das Routing der Anfrage in Abhängigkeit von dem Format der Anfrage zu bestimmen. Dies Ermöglicht es für eine Anfrage die Action innerhalb eines ActionControllers auszuwählen, welche entwickelt wurde um eine Anfrage in diesem Format zu verarbeiten. Ist in der Routingtabelle zu einer Anfrage in einem bestimmten Format keine Action hinterlegt, so wird die Anfrage dementsprechend zurückgewiesen. Prinzipiell bietet es sich an, den gesamten Funktionsumfang einer API Schnittstelle innerhalb eines dedizierten ActionControllers umzusetzen. Nach dem seperation of concerns Programmierparadigma reduziert dies einerseits durch Entkopplung die Gefahr einer Unbeabsichtigte Änderung der Funktionsweise, andererseits erhöht sich hierdurch auch die Wartbarkeit der Schnittstelle. Die Bündelung des gesamten Funktionsumfangs der API Schnittstelle innerhalb eines ActionControllers birgt noch einen weiteren Vorteil in Bezug auf die Versionierung der API Schnittstelle. Dadurch, dass ein ActionController für die Umsetzung des vollen Funktionsumfangs der API zuständig ist, muss bei der Entwicklung der nächsten Version der API nicht mehr getan werden, als einen neuen ActionController auf Basis der letzten Version zu erstellen und diesen um die neue Funktionalität zu erweitern. Dies hat selbstverständlich den Nachteil, Redundanz innerhalb des Systems zu Schaffen und damit gegen das Prinzip von don't repeat yourself zu arbeiten. Dieser Nachteil wird aber Aufgewogen durch die Stabilität und unabhängige Wartbarkeit der einzelnen API Versionen, welche durch die strikte Entkoppelung zwischen den Versionen entsteht. Damit der Dispatcher der Rails Anwendung bei einem Aufruf der API entscheiden kann, welche Version der API angefragt wird, und damit auch an welchen ActionController der Aufruf delegiert werden soll, ist es mit Rails möglich innerhalb der Routingtabelle namespaces [ROR10] zu verwenden. Innerhalb des API namespaces kann für jede existierende Version der API ein untergeordneter namespace angelegt werden. Bei einem Aufruf einer Funktion der API muss der konsumierende Client die Version der API innerhalb in der URI angeben.

3.2.5 ActionMailer und ActiveJobs

Mit Hilfe der Klasse ActionMailer [ROR7] ist es einer Rails Anwendung möglich, E-Mails zu versenden. Der Aufbau des ActionMailers erinnert hierbei an den Aufbau des ActionControllers. Eine Action des ActionMailers ist für die Generierung und den Versand einer E-Mail zuständig. Mit Hilfe von Klassenvariablen des ActionMailers, übergebenen Parametern und Zugriff auf die ActiveRecords bereitet er, innerhalb eines Methodenaufrufs, alle für die E-Mail relevanten Daten, wie zum Beispiel

Absender, Empfänger, Betreff vor und delegiert den Aufruf an eine ActionView, welcher die E-Mail mit Hilfe eines Template erzeugt.
Seit Rails Version 4.2 wird für den Versand von E-Mails, die durch den ActionMailer generiert wurden das ActiveJob Framework [ROR8] benutzt. Dieses Framework erzeugt durch die Serialisierung aller relevanten Informationen, die für den Versand einer E-Mail benötigt werden einen ActiveJob. Ein ActiveJob erlaubt es, den Versand der serialisierten E-Mail unabhängig vom Rest des Systems zu einem beliebigen Zeitpunkt auszuführen. Für die Verarbeitung von ActiveJobs bietet das Framework Adapter für eine Reihe von Message Queues Backends [ROR9], die es unter Anderem möglich machen E-Mails priorisiert, Zeitplan gesteuert und asynchron zur Rails Anwendung zu versenden.

3.2.6 Optimierung der Benutzerfreundlichkeit der Weboberfläche

Um mit einem HTML basierten Webdienst den Erwartungen heutiger Benutzer in Bezug auf Präsentation und Benutzerfreundlichkeit einer Website gerecht zu werden, kommt man nicht um die Nutzung von CSS und JavaScript herum. CSS Style Vorgaben erlauben es, eine einheitliche und insgesamt stimmige Präsentation von Webseiten zu gestalten.
Neben der Präsentation ist auch die Handhabung einer Website von entscheidender Bedeutung. Für die Optimierung der Handhabung und damit die Verbesserung der Benutzererfahrung in Bezug auf eine Website ist der Einsatz von JavaScript ein geeignetes Mittel. Ein Seitenaufruf einer klassischen Website besteht darin, dass der Browser eine Verbindung zum Server der Seite aufbaut, dort das HTML Dokument herunterlädt und als erstes den Inhalt des HEAD Tags der HTML Seite liest. Besitz der HEAD Tag Verweise zu CSS oder JavaScript Dateien, so werden diese erst heruntergeladen. Im Ausführung wird der JavaScript Code ausgeführt und erst danach fängt der Browser mit dem Rendern der eigentlichen Seite anhand der HTML und der CSS Datei an. Auch, wenn heutige Browser den Ladevorgang von CSS und JavaScript Dateien durch Cache Mechanismen optimieren, so bleibt die Ausführung von JavaScript eine synchrone und damit blockierende Prozedur, welche je nach Größe der Seite und Latenz der Verbindung zwischen Browser und Server negativ auffallen kann. Seit der Spezifikation von HTML 5 [HTM5] ist es möglich JavaScript Code durch das setzen des async Attributes [ASYN] asynchron zu laden und auszuführen. Hierdurch wird das Rendern einer Website nicht mehr blockiert und sie wird schneller vom Browser aufgebaut.
Um den Cache Mechanismus von Browsern in Bezug auf CSS und JavaScript Dateien optimal zu nutzen, bietet es sich an sämtliche Style Vorgaben und sämtlichen

JavaScript Code einer Website in jeweils einer CSS beziehungsweise einer JavaScript Datei zu bündeln. Hierdurch muss der Browser bei Folgeaufrufen innerhalb der Website nur noch das HTML Dokument und möglicherweise Bilddateien laden. Rails folgt dieser Herangehensweise und kompiliert für den Produktivbetrieb einer Anwendung sämtliche CSS und JavaScript Anteile der Anwendung in jeweils eine Datei.
Das JavaScript Framework Turbolinks[TURB], welches seit Version 4.0 fester Bestandteil von Rails ist, greift diesen Ansatz auf und führt ihn noch einen Schritt weiter. Da dem Browser nach dem ersten Aufruf sowohl alle Style Vorgaben und der gesamte JavaScript Code einer Rails Anwendung vorliegt, ist es eigentlich nicht nötig, diese bei jedem Folgeaufruf erneut zu parsen beziehungsweise auszuführen. Auf dieser Tatsache basierend fängt Turbolinks alle Folgeaufrufe innerhalb einer Rails Anwendung mit Hilfe einer JavaScript Funktion ab. Diese Funktion ersetzt das Standardverhalten des Browsers und lädt nur den Bereich des HTML Dokuments, welcher durch das BODY Tag eingeschlossen wird vom Server und ersetzt diesen Bereich innerhalb des aktuellen HTML Dokuments des Browsers. Hierdurch fällt ein erneutes Parsen der CSS Style Vorgaben und ein erneutes Ausführen des JavaScript Codes weg und die Seitenaufrufe werden beschleunigt.
Da es möglich ist, mit Hilfe von JavaScript jedes Element innerhalb der Baumstruktur des Data Object Models [W3DO] eines HTML Dokumentes zu manipulieren, lässt sich der beschriebene Ansatz von Turbolinks noch weiter ausbauen. Verändert sich bei einem Aufruf nur ein Teilbereich des aktuell vom Browser dargestellten Dokuments, so ist nicht nötig den gesamten Inhalt innerhalb des BODY Tags neu zu laden. Hier bietet es sich an, eine JavaScript Funktion zu entwerfen, welche bei einem solchen Aufruf nur den Teil der Seite vom Server anfragt, welcher ersetzt werden muss und das aktuell dargestellte Dokument aktualisiert. Auf dieselbe Weise ist es möglich, bei dem initialen Aufruf einer Seite nur den Bereich der Seite zu laden, welcher von dem Benutzer unmittelbar zu sehen ist und erst beim Scrollen der Seite durch den Benutzer weiteren Inhalt dynamisch nachzuladen. In all diesen Ansätzen ist es wichtig, dass die Funktionalität zum Laden partieller Inhalte sowohl clientseitig anhand des JavaScript Codes als auch serverseitig durch den Webdienst implementiert wird.
Sowohl die Präsentation, als auch die Handhabung und Antwortzeiten einer Website haben einen nicht zu unterschätzenden Einfluss auf die Wahrnehmung der Qualität eines Webdienstes durch einen Benutzer. In dem Kapitel Realisierung wird das Verfahren des partiellen Ladens von Inhalten anhand eines Beispiels demonstriert.

3.3 Entwicklung mit dem Android SDK

Die Komponente des Systems für die Registrierung der Betriebsmittel für den Verleih, die Reservierung und die Rückgabe wird für ein mobiles Endgerät entwickelt, auf dem das Android Betriebssystem läuft. Um den Rahmen dieser Abschlussarbeit nicht zu sprengen, geht es in diesem Abschnitt nicht um die Systemarchitektur des Android Betriebssystems im Ganzen, sondern um die Anwendungsschicht und den Anwendungsrahmen aus Sicht der Entwicklung einer Android Applikation im speziellen.

3.3.1 Anwendungsschicht und Anwendungsrahmen

Die Anwendungsschicht ist der Bereich im Android Betriebssystem, in welchem Anwendungen ausgeführt werden und die Interaktion einerseits zwischen Benutzer und Anwendung, andererseits zwischen verschiedenen Anwendungen untereinander, gesteuert wird. Hierbei setzt die Anwendungsschicht auf den Anwendungsrahmen auf, welcher in Form des Android SDK eine Vielzahl von Bibliotheken, Diensten und auf Java basierenden Klassen für die Implementierung von Anwendungen anbietet. [BECK15]

Mit Hilfe des Android SDK ist es möglich, über Systemdienste Zugriff auf die Hardwarekomponenten eines Android Gerätes zu bekommen, wie zum Beispiel den GPS Empfänger oder die Kamera. Als Basis für die Umsetzung von Android Applikationen bietet das SDK eine Auswahl von Klassen für die Lösung gängiger Probleme an. Zum Beispiel einen HTTP Client für den Kommunikationsaufbau zu HTTP basierten Webdiensten oder eine Klasse zur Erstellung von asynchronen Aufgaben innerhalb einer Android Applikation.

Die vom Android SDK bereitgestellte Klasse Context und mit ihr die ableitenden Klassen Activity und Service bilden den Ausführungsrahmen einer Android Applikation.

3.3.2 Die Klasse Context

Die Klasse Context ist die zentrale Komponente für die Umsetzung der Anwendungslogik und für die innerhalb der Anwendungsschicht. Sie dient als Aufrufender Kontext für alle Operationen einer Android Applikation.

Von der Klasse Context leiten sich zwei Klassen ab, Activity und Service. Der Hauptunterschied zwischen ihnen liegt darin, das Activity für die direkte Interaktion mit dem Benutzer konzipiert ist und hierfür eine Darstellungsebene in Form von

Views anbietet, während der Service für die Umsetzung von Prozessen entwickelt wurde, welche im Hintergrund laufen und keine direkte Interaktion mit dem Benutzer erfordern. Am Beispiel eines E-Mail Clients würde die Oberfläche und die Interaktion mit der Android Applikation mit Hilfe einer Activity umgesetzt werden. Das periodische Abrufen der E-Mails vom Server könnte hingegen von mit einem Service im Hintergrund gelöst werden.
Bei der Präsentationsebene einer Android Applikation, so wie sie mit Hilfe der Activity umgesetzt werden kann, wird nicht strikt zwischen Viewer und Controller unterschieden. Das Android SDK bietet zwar in Form von XML Templates die Möglichkeit Views von der Anwendungslogik der Activity zu kapseln, prinzipiell ist es aber auch möglich Views innerhalb von Methoden der Activity programmatisch zu erzeugen und zu manipulieren.

3.3.3 Die Rolle von dem Android SDK innerhalb des Systems

Die Android Applikation, die mit Hilfe des Android SDK im Rahmen dieser Arbeit erstellt wird, soll am Lagerort der Betriebsmittel einen Zugriffspunkt für Benutzer zum System bieten. Mit Hilfe der Kamera soll es ermöglicht werden, Betriebsmittel für die weitere Verarbeitung innerhalb des Systems zu registrieren. Hierfür muss die Android Applikation in der Lage sein, über das unternehmensinterne Netzwerk mit der Rails Anwendung zu interagieren.

4 Konzept

In diesem Kapitel werden ausgewählte Teile der Konzeptionierung des Gesamtsystems vorgestellt. Anhand der Systemarchitektur wird ein Überblick über die wichtigsten Komponenten gegeben und deren Position innerhalb des Systems gezeigt. Im Abschnitt Modelle werden die wichtigsten Komponenten der Modellebene und deren Relation zueinander vorgestellt und am Beispiel des Models Item wird das Konzept der STI erläutert.

4.1 Systemarchitektur

Das System besteht aus zwei Teilsystemen. Zum einen eine Rails Anwendung, welche mit dem Anwendungskern die Funktionalität des Systems umsetzt und im Sinne eines Servers über verschiedene Schnittstellen nach außen anbietet. Zum anderen eine Android Applikation, welche als entfernte Komponente für die Registrierung von Betriebsmitteln zur Verarbeitung innerhalb des Systems zuständig ist. Die Interaktion zwischen diesen beiden Teilsystemen erfolgt über eine API Schnittstelle an. Das System ist nach dem MVC Pattern aufgebaut.

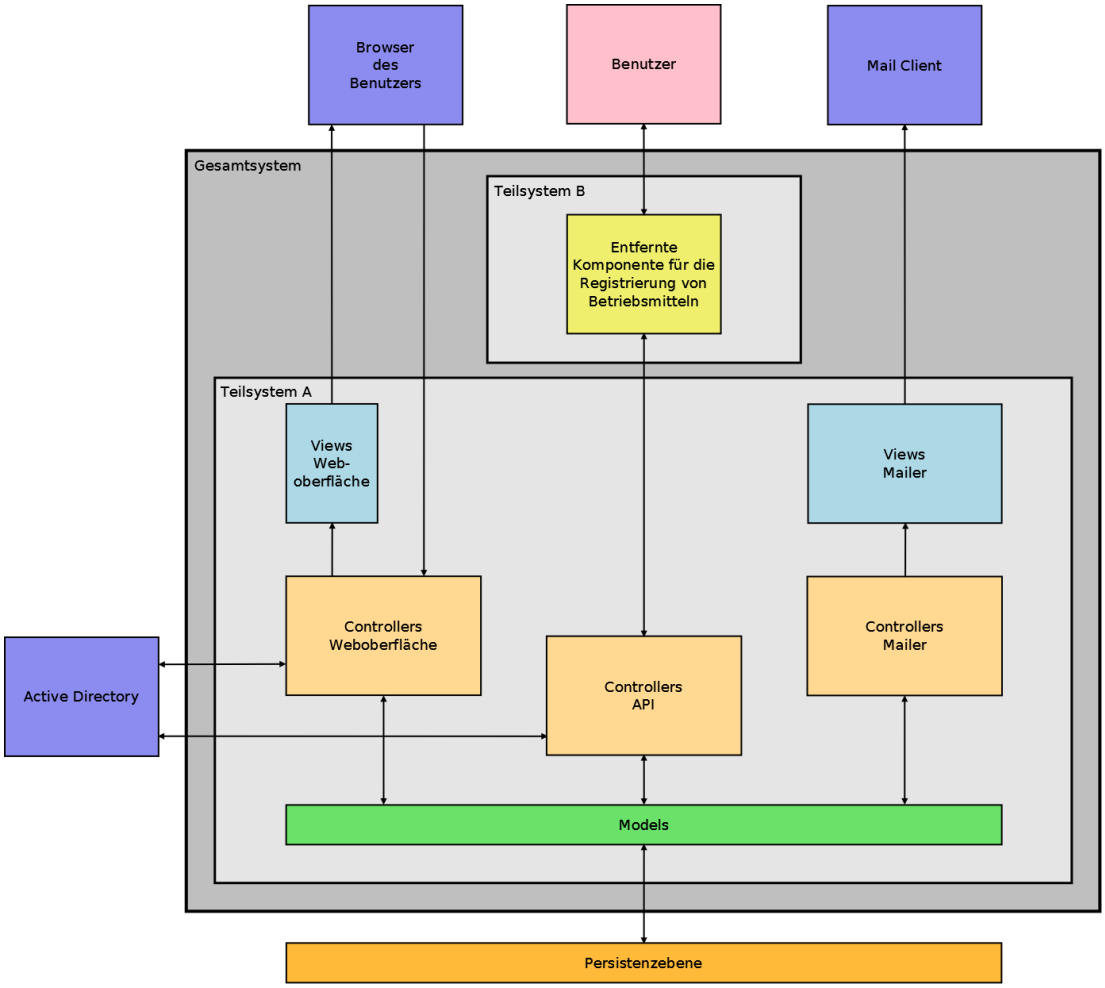

Abbildung 2 - Systemarchitektur

Wie in der Abbildung 2 - Systemarchitektur zu sehen, lassen sich die Controller, anhand der Dienste, die sie innerhalb des Systems, aber auch nach außen hin anbieten, in 3 Bereiche einteilen. Eine Teilmenge der Controller ist zuständig für Interaktion zwischen System und Benutzern über die Weboberfläche, ein weiterer Teilbereich für die Interaktion zwischen entfernter Komponente und Anwendungskern über die API Schnittstelle und der letzte Teilbereich für den Versand von Benachrichtigungen an die Benutzer in Form von E-Mails. Das System besitzt eine Anbindung an das externe Active Directory System des Unternehmens. Hierdurch ist es möglich, Benutzer als Mitarbeiter des Unternehmens zu authentifizieren und für das System relevante Daten zu diesen Benutzern abzufragen, wie zum Beispiel den vollständigen Namen oder die E-Mail-Adresse. Um die

Abbildung nicht zu Überladen, wurde an einigen Stellen stark abstrahiert. Auch wurde auf die Darstellung der inneren MVC Struktur des Android basierten Teilsystem für die Registrierung von Betriebsmitteln verzichtet, um die Abbildung nicht zu unübersichtlich zu gestalten. Abgesehen von technischen Modellen gibt es innerhalb des Systems 10 Modelle, auf welche in den folgenden Abschnitten teilweise noch näher eingegangen wird. Zu fast allen Modellen existiert ein dazugehöriger Controller für die Verarbeitung von Anfragen in Bezug zu dem Modell und jeder dieser Controller besitzt wiederum eine Vielzahl von Views für die Präsentationsebene und Interaktion zwischen System und Benutzer.

4.2 Modelle

Dieser Abschnitt vermittelt einen Überblick über alle wichtigen Modelle und geht im speziellen auf die Beziehungen zwischen ihnen ein. Hierbei besitzen beide Teilsysteme eine eigene Repräsentation der Teilmenge von Modellen, die sie für die Umsetzung ihres jeweiligen Funktionsumfangs benötigen.

4.2.1 Relationen zwischen den Modellen

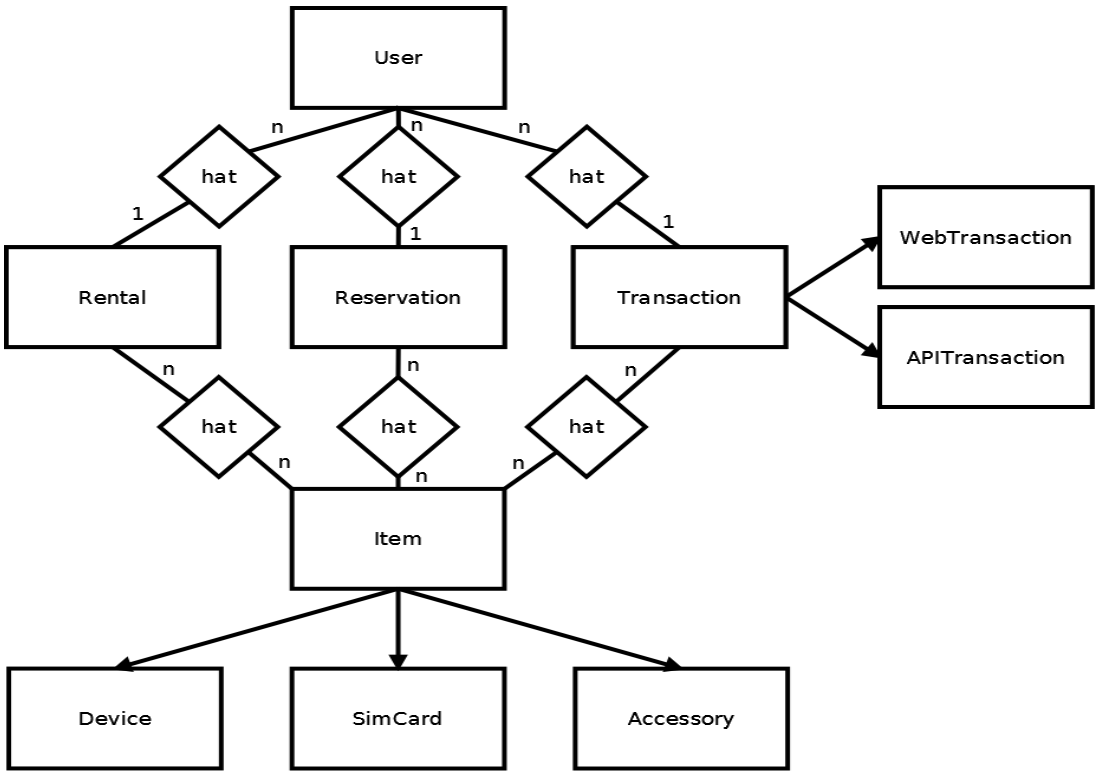

Abbildung 3 - Vereinfachtes ER Model

Die Abbildung 3 - Vereinfachtes ER Model zeigt einen Überblick über die Beziehungen zwischen den Modellen, wobei für einen besseren Überblick auf die Darstellung der Attribute der Entitäten und auf rein technische Entitäten verzichtet. So besitzt zum Beispiel die Beziehung zwischen Reservation und Item zwei weitere Ebenen an rein technischen Entitäten. Auf die Gründe und die Struktur wird im folgenden Abschnitt genauer eingegangen.
Die Pfeile zwischen Entitäten macht die Vererbungshierarchie zwischen ihnen deutlich. Hierbei gelten die Beziehungen einer Basisklasse zu anderen Entitäten implizit auf für alle ableitenden Klassen.

4.2.2 Beziehung zwischen Reservation und Item im Detail

Die Struktur zwischen Items auf der einen Seite und Transactions, Reservations und Rentals auf der anderen Seite entspricht einer n:n Beziehung und ist auf der Datenbankebene nicht direkt abzubilden. An dieser Stelle wird am Beispiel von Reservation und Item gezeigt, wie eine Struktur zwischen diesen Entitäten entworfen werden muss, um eine Abbildung auf die Datenbank zu ermöglichen.

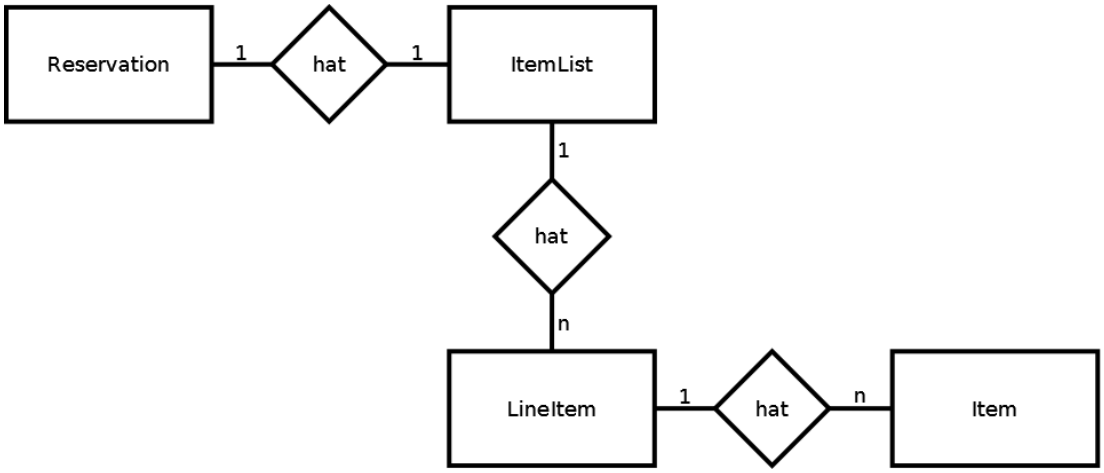

Abbildung 4 - ER Model zwischen Reservation und Item

Wie in der Abbildung 4 - ER Model zwischen Reservation und Item zu sehen ist, befinden sich zwischen Reservation und Item zwei weitere rein technische Entitäten. Sie dienen als Hilfsmittel und erfüllen unterschiedliche Aufgaben innerhalb der Beziehung zwischen Reservation und Item. Die Entität LineItem dient als Hilfsmittel, um die n:n Beziehung auf Ebene der Datenbank darstellen zu können. Ein Datensatz eines LineItems referenziert hierbei ein Item und, über die ItemList, eine Reservation und stellt damit die Zugehörigkeit eines Items zu einer bestimmten Reservierung dar. Da zu einer Reservation beliebig viele LineItems existieren können und auch jedes Item von beliebig vielen LineItems referenziert werden kann, ermöglicht dies die Beziehung zwischen Reservation und Item auf Datenbankebene darzustellen.
Sieht man von der ItemList ab erst einmal ab, so würden die Datensätze von Reservation, Item und dazugehörigem LineItem folgendermaßen aussehen:

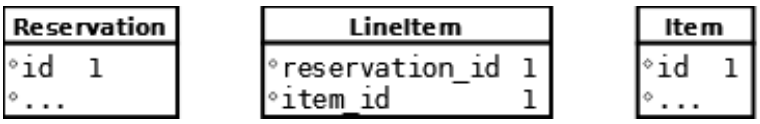

Abbildung 5 – Vereinfache Beziehung zwischen Transactions und Items mit LineItems

Innerhalb des LineItems wird in diesem Beispiel die Reservation direkt referenziert. Damit für Rental und Transaction eine analoge Datenstruktur geschaffen werden kann gibt es mehrere Ansatze.
Die eine Möglichkeit besteht darin drei Typen von LineItems zu erzeugen, ein ReservationLineItem, ein RentalLineItem und ein TransactionLineItem. Dies führt dazu, dass statt einer Tabelle drei erzeugt werden müssen. Die zweite Möglichkeit besteht darin, drei Felder innerhalb von LineItem zu erstellen und anhand des Feldes die Zugehörigkeit zu speichern. Dies würde bedeuten, dass immer nur eins der drei referenzierenden Felder belegt sein darf und ist sehr fehleranfällig. Außerdem führen Beide ansätze zu einer sehr starken Kopplung zwischen der Implementierung der LineItems und Reservation, Rental und Transaction.
Dieses Problem wird mit Hilfe der Entität ItemList gelöst.

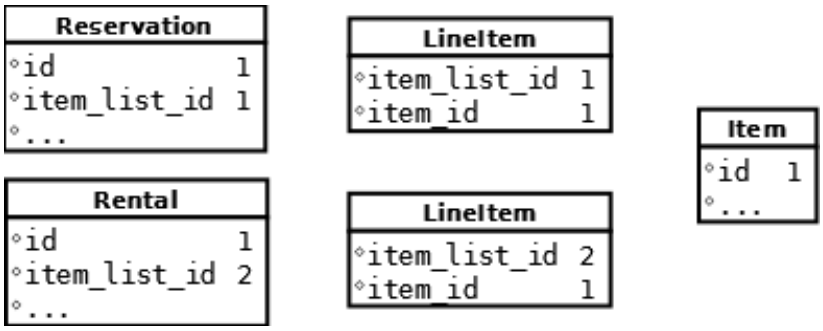

Abbildung 6 - Beziehung zwischen Transactions und Items mit ItemList

Wie man in Abbildung 6 - Beziehung zwischen Transactions und Items mit ItemList sieht, referenzieren nun sowohl die einzelnen Transaktionstypen als auch die LineItems eine abstrakte ItemList. Hierdurch ist das LineItem komplett von den drei Typen von Transaktionen entkoppelt.

4.2.3 Wahl von STI oder MTI am Beispiel Item

In dem Abschnitt Grundlagen wurde bereits auf die Konzepte STI und MTI zur Realisierung von Vererbung innerhalb von Rails Anwendungen eingegangen. In diesem Abschnitt wird am Beispiel des Modells Item und den ableitenden Modellen Device, SimCard und Accessory die Wahl des Konzeptes begründet und auf wichtige Details bei der Konzeptionierung der Vererbung unter diesen Modellen innerhalb des Systems eingegangen.
Um eine Entscheidung treffen zu können, müssen zwei Aspekte betrachtet werden. Zum einen müssen die Zugriffe innerhalb des Systems auf diese Modelle untersucht

werden und zum anderen die erwartete Auswirkung auf die Größe der Datenbank in Abhängigkeit von der Wahl des Konzeptes.
Wie das ER-Model aus dem letzten Abschnitt deutlich macht, verwalten die Entitäten Rental, Reservation und Transaction die Entität Item direkt. Die Operationen dieser Entitäten beziehen sich alle auf die Gesamtheit der von ihnen verwalteten Items und nicht auf die Teilmengen von Devices, SimCards oder Accessories. Für die Datenbankabfragen, die im Zuge dieser Operationen erzeugt werden, ist es also notwendig auf dem gesamten Datensatz von Items zu arbeiten. Da, wie in Grundlagen beschrieben, hierfür bei dem Konzept der MTI bei jeder Abfrage eine vergleichsweise rechenintensive FULL INNER JOIN Operation der Tabellen aller ableitenden Klassen notwendig ist, während bei dem Konzept STI bereits eine Tabelle für alle Items zur Abfrage vorliegt, ist in diesem Zusammenhang STI die eindeutig performantere Wahl. Als zweiter Aspekt muss betrachtet werden, wie sich die Wahl des Konzeptes auf die erwartete die Größe der Daten innerhalb der Datenbank auswirkt.

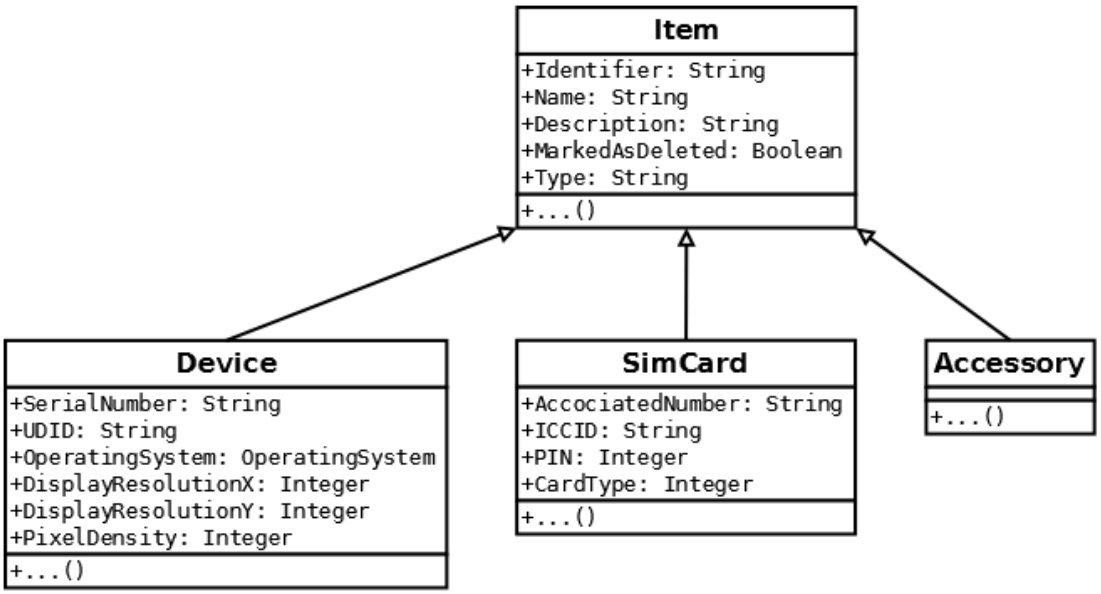

Abbildung 7 - UML Diagramm Item und ableitende Klassen

Mit Hilfe der Abbildung 7 - UML Diagramm Item und ableitende Klassen wird gezeigt, welche Attribute persistent in der Datenbank gehalten werden müssen. Ein Entwurf dieser Vererbungshierarchie mit STI würde bedeuten, dass für einen Datensatz, egal ob es sich hierbei um ein Device, eine SimCard oder ein Accessory handelt innerhalb der Tabelle die Summe aller Attribute hinterlegt sind. Bei einem Device sind das 4 nicht benötigte Attribute, bei einer SimCard 6 und bei einem Accessory sogar 10. Dies

bedeutet, auch wenn diese unnötigen Attribute innerhalb der Tabelle mit null versehen werden, einen größeren Platzbedarf innerhalb der Datenbank als bei dem MTI. Betrachtet man das Verhältnis von Typen von Betriebsmitteln zueinander, so wie sie im Unternehmen bei der Umsetzung dieses Projektes vorliegen, so überwiegt die Anzahl von Devices den von SimCards und Accessories derart stark, dass im Durchschnitt knapp über 5 Felder pro Datensatz nicht typbezogen wären.
Bei der Abwägung zwischen dem Gewinn an Leistung bei Abfragen mit STI und dem geringeren Platzbedarf mit MTI ist die Wahl auf STI gefallen. Grund hierfür ist vor allem, dass die Anzahl von Items im Laufe der Zeit sehr viel langsamer steigt, als die Anzahl von Rentals, Reservations und Transactions. Hierdurch überwiegt der Gewinn von Leistung im Hinblick auf Datenbankabfragen bei weitem den größeren, aber insgesamt immer noch relativ geringen Wachstum der Datenmenge.
Für die Konzipierung der Vererbungshierarchie werden die Attribute der Basisklasse Item und die Attribute der ableitenden Klassen nun also in der abstrakten Klasse Item gebündelt. Auch wenn Item de facto alle Attribute besitzt und an die ableitenden Klassen vererbt, so erfolgt die die Validierung der Attribute, nach dem Prinzip seperation of concern, innerhalb der Klasse, zu der das Attribute gehört. Hierbei wird die Validierung von Attribute, die alle ableitenden Klassen gemeinsam haben, in der Basisklasse Item durchgeführt. Dank der Hilfsmethoden von Rails unterscheiden sich Methodenaufrufe auf die so konzipierten ableitenden Klassen nicht von Aufrufen auf Klassen mit Abbildung auf eine eigene Tabelle und es müssen bei ihnen keine weiteren Maßnahmen getroffen werden.

4.3 Controller

Das System besitzt eine Vielzahl von Controllern um die Interaktion zwischen System und Außenwelt, aber auch zwischen den Teilsystemen zu realisieren. Dabei gibt es modellspezifische Controller, welche den Funktionsumfang des Systems mit Bezug auf ein bestimmtes Modell umsetzen und es gibt Controller, welche eine Menge von zusammengehörender Funktionalität des Systems umsetzten. Ein Beispiel hierfür ist der APIController, welcher den Funktionsumfang für die Schnittstelle zwischen beiden Teilsystemen umsetzt. In diesem Abschnitt wird am Beispiel des ItemControllers die Konzipierung eines modellspezifischen Controllers gezeigt und am Beispiel des APIControllers die eines schnittstellenspezifischen Controllers.
Da beide Controller Bestandteil des mit Hilfe von Rails entwickelten Anwendungskerns sind, weisen sie zum Teil Rails spezifische Ansätze zur Problemlösung auf.

4.3.1 ItemController

Der ItemController ist für die Umsetzung der Anfragen zuständig, die ein Benutzer über die Weboberfläche in Bezug auf Items anstoßen kann. Da die Klasse Item abstrakt ist, beziehen sich die Methoden des Controllers grundsätzlich auf eine der drei ableitenden Klassen. Um das Verhalten der auf die konkreten Ableitende Klasse abstimmen zu können, muss diese in Form eines Typs bei einer Anfrage mit angegeben werden. Dies wird mit Hilfe der Routingtabelle gelöst. In ihr liegt zu jeder ableitenden Klasse jeweils ein Satz aller verfügbaren Aufrufe in Form von Routen vor. Hierdurch kann der Dispatcher anhand des Pfades der URL entscheiden, an welche ableitende Klasse der Aufruf gerichtet ist und übergibt diese Information beim delegieren des Aufrufs mit an den ItemController.

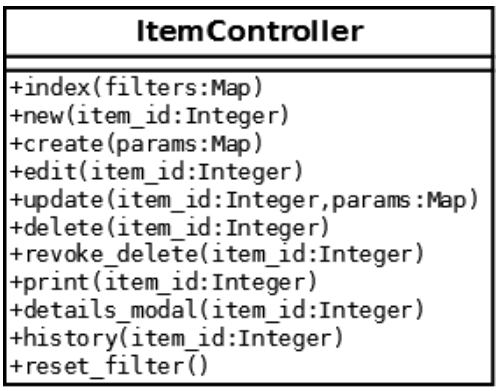

Abbildung 8 - UML Diagramm des ItemControllers

In der Abbildung 8 - UML Diagramm des ItemControllers ist ersichtlich, welche Methoden der ItemController anbieten soll, um Anfragen über die Weboberfläche zu verarbeiten. Bei der Benennung der Methoden wird auf die übliche Namenskonvention von Rails geachtet. Die index Methode soll einen Überblick aller Items des übergebenen Typs anzeigen. Optionale Filterparameter sollen dabei die Auswahl der angezeigten Items weiter einschränken können. Mit Hilfe der reset_filter Methode können die Filteroptionen wieder auf den ursprünglichen Zustand zurückgesetzt werden.

Die Methoden-Paare new und create beziehungsweise edit und update sind für die Erzeugung eines neuen Items beziehungsweise der Änderung eines übergebenen Items zuständig. Bei der Methode new soll es darüber hinaus möglich sein, als optionalen Parameter ein bereits existierendes Item zu übergeben. In diesem Fall dienen die Attribute des übergebenen Items als Basis für das Erstellungsformular eines neuen Items.

Da Items nicht komplett aus dem System gelöscht werden sollen, werden diese bei einem Aufruf von delete nur als gelöscht markiert. Markierte Items stehen nicht mehr für Transaktionen zur Verfügung und sind nur noch für administrative Benutzer einsehbar. Für sie bleiben alle Informationen des Items erhalten und es ist ihnen weiterhin möglich mit Hilfe der history Methode Protokolle über vergangene Transaktionen einzusehen. Diese Markierung soll bei Bedarf über revoke_delete wieder Rückgängig gemacht werden können, womit das Item wieder für Transaktionen zur Verfügung steht.
Über die Print Methode soll es dem Benutzer möglich sein den Druck des QRCodes anzustoßen, welcher für die Registrierung der Betriebsmittel mit Hilfe der Android Applikation benötigt wird.
Mit Hilfe der Methode details_modal soll der Benutzer detaillierte Informationen zu einem Item erhalten. Um die Usability auf der Präsentationsebene zu verbessern, soll der Browser des Benutzers bei diesem Aufruf nicht von der aktuellen Seite auf eine neue Seite wechseln, sondern die Informationen in einem kleinen Fenster modal auf der aktuellen Seite anzeigen. Dieses Problem wäre prinzipiell mit HTML und damit innerhalb des Views lösbar und würde keine Interaktion mit dem Controller verlangen. In diesem Fall liegen die Informationen aller angezeigten Items bereits innerhalb des HTML Dokuments der aktuellen Seite in versteckter Form vor und werden bei Bedarf für ein bestimmtes Item durch angezeigt. Dieser Ansatz hat aber einen entscheidenden Nachteil. Selbst bei einer kleinen Übersicht von Items müssen dem HTML Dokument zu jedem Item bereits beim Aufruf der Übersicht alle Informationen vorliegen. Dies bedeutet im Vergleich zu der Generierung der Übersicht der Items ohne diese Informationen einen nicht zu unterschätzenden Mehraufwand, der sich bei der Arbeit mit dem System stark bemerkbar machen würde. Wie in den Grundlagen beschrieben lässt sich dieses Problem mit Hilfe von JavaScript lösen. Hier wird nur bei Bedarf der HTML Schnipsel mit den Informationen zu dem Item dynamisch in das HTML Dokument nachgeladen. Damit dies überhaupt möglich ist, muss der ItemController einen Aufruf anbieten, welcher diesen HTML Schnipsel für die JavaScript Funktion generiert.

4.3.2 Konzept der Interplattform-Kommunikation

Am Beispiel des APIControllers wird im Folgenden das Konzept der serverseitigen Umsetzung der Schnittstelle für die Interaktion zwischen der Rails Anwendung und der Android Applikation zur Registrierung von Betriebsmitteln für den Verleih, die Reservierung oder die Rückgabe demonstriert.

Grundsätzlich wird bei allen Aufrufen an den APIController eine Antwort in Form einer JSON Struktur [JSON] generiert. Fester Bestandteil dieser Struktur ist ein Feld, welches Auskunft über den Erfolg der Operation gibt. Abhängig von dem Erfolg der Operation existiert innerhalb der Struktur entweder eine Sektion mit einem message Objekt oder eine Sektion mit ein bis mehreren error Objekten beinhalten. In diesen Objekten werden genaue Informationen über das Resultat der Operation aufgeführt, sowohl in für Maschinen als auch in für Menschen lesbarer Form. Zusätzlich kann eine Antwort abhängig von der Operation weitere Informationen in einer Payload Sektion innerhalb der JSON Struktur besitzen.

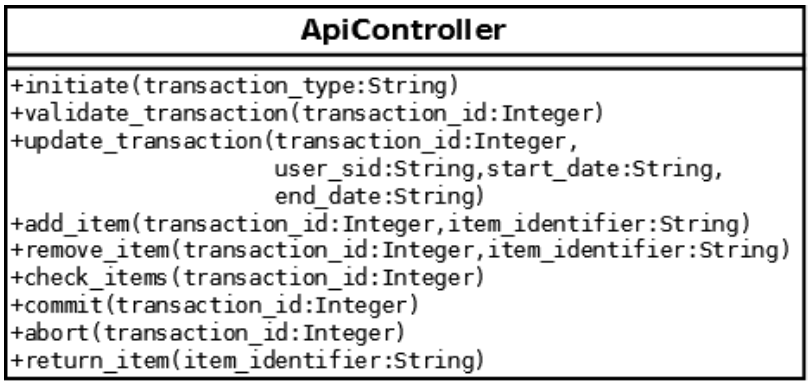

Abbildung 9 - UML Diagramm des APIControllers

Die Abbildung 9 - UML Diagramm des APIControllers bietet einen Überblick über die Methoden des APIControllers, welche der Android Applikation über HTTP Aufrufe zur Verfügung stehen. Das Reservieren und das Ausleihen der Betriebsmittel erfolgt innerhalb dabei nach dem Schema einer Transaktion.
Hat sich der Benutzer innerhalb der Android Applikation entschieden, dass er Betriebsmittel leihen oder reservieren möchte, wird durch einen Aufruf der initate Methode unter Angabe des ausgewählten Transaktionstyps eine Transaktion erstellt und der Android Applikation innerhalb der Antwort eine Referenz für die Transaktion mitgeteilt. Diese muss bei weiteren transaktionsbezogenen Operationen angegeben werden. Während die Methode update_transaction die Informationen über die Rahmenbedingungen der Transaktion zu übertragen, erlauben die Methoden add_item und remove_item das Hinzufügen und Entfernen von Items. Um den Prozess für die Mitarbeiter so flexibel wie möglich zu gestalten, ist die Reihenfolge dieser drei Operationen nicht vorgegeben. Es ist auch nicht zwingend notwendig alle Parameter von update_transaction in einem Aufruf zu übergeben. Erst bei dem Aufruf von check_items oder von commit wird auf das Vorhandensein aller benötigten Informationen für die Transaktion geprüft. Fehlende Informationen

werden der Android Applikation im Rahmen der Antwort mitgeteilt. Die Methode check_items ermöglicht es der Android Applikation die Verfügbarkeit der Betriebsmittel in dem vom Benutzer angegebenen Zeitraum zu überprüfen. Mit Hilfe von commit kann wird der Abschluss der Transaktion angefordert. Die Methode abort erlaubt es schließlich die Tranksaktion bei Bedarf abzubrechen.
Um auch die Rückgabe für den Benutzer so flexibel wie möglich zu gestaltet, soll es einerseits möglich sein, nur eine Teilmenge von Items aus einer Ausleihtransaktion wieder zurück zu geben, andererseits soll es auch möglich sein, Items von verschiedenen Rentals in einer beliebigen Reihenfolge zurück zu geben. Für die Rückgabe soll es nicht notwendig sein, dass sich der Benutzer am System anmeldet. Aus diesem Grund ist die Rückgabe, welche durch die return_item Methode aufgerufen kann, eine für jedes Item in sich abgeschlossene Operation.

5 Realisierung

In diesem Kapitel werden wichtige Aspekte bei der Realisierung des Systems vorgestellt und einige Teile der Implementierung sowohl der Interplattform Kommunikation, als auch der Optimierung der Reaktionszeit des Systems gezeigt.

5.1 Testgetriebene Entwicklung

Für die Implementierung des Anwendungskerns in Rails wurde nach den Konzepten der test- und der verhaltensgetriebenen Entwicklung vorgegangen. Für die Erstellung der Tests wurde das für Rails entwickelte Testframework RSpec [RSPE] benutzt. Dieses Framework erlaubt es, zentrale Aspekte der verhaltensgetriebenen Entwicklung umzusetzen. So ist zum Beispiel die Benennung sowohl von Testfällen, als auch die Beschreibung des erwarteten Verhaltens innerhalb der Testfälle, in einer natürlich sprachlichen Form möglich.

```ruby
describe Api::V1::BaseController, :type => :controller do
  describe "trying to rent" do
    # ...
    it "by adding a device, which does not exist in the system" do
      # Initiate Rental-Transaction
      post :initiate, type: type, format: :json
      response_to_hash = ActiveSupport::JSON.decode(response.body)
      expect(response_to_hash["success"]).to eq(true)
      transaction_id = response_to_hash["transaction"]["id"]

      # Add Device that does not exist to rental, using the TransactionID
      invalid_identifier = "InvalidIdentifier"
      post :add_item, transaction_id: transaction_id, identifier: invalid_identifier, format: :json
      response_to_hash = ActiveSupport::JSON.decode(response.body)
      expect(response_to_hash["success"]).to eq(false)
      error_statuses = response_to_hash["errors"].collect { |error| error["status"] }
      expect(error_statuses).to include("item_not_found")
    end

    it "by adding two Devices" do
      # ...
    end

    it "by adding a Device which is already part of the rental" do
      # ...
    end

    it "by adding two Devices, then deleting one from the transaction" do
      # ...
    end
  end
end
```

Abbildung 10 - Kaskadierende Testfallbeschreibung

Die Abbildung 10 - Kaskadierende Testfallbeschreibung zeigt am Beispiel von Testfällen für den APIController die Form, in der Testfälle in RSpec erstellt werden können. Mit Hilfe dieser Struktur wird bei der Auswertung von Fehlern deutlich, welches erwartete Verhalten vom System nicht erfüllt wurde.

5.2 Interplattform Kommunikation

Die Interaktion zwischen der Rails Anwendung und der Android Applikation hat zwei zentrale Aspekte. Auf der einen Seite den Aufruf innerhalb der Android Applikation auf die API Schnittstelle, und auf der anderen Seite die Verarbeitung und Beantwortung des Aufrufs durch die Rails Anwendung. In diesem Abschnitt wird am Beispiel des Hinzufügens eines Items zur aktuellen Transaktion die Implementation einer solchen Interaktion gezeigt.

a) Scan des Testgeräts

Um innerhalb der Android Applikation QRCodes einscannen und auswerten zu können, wird die Java Bibliothek Zebra Crossing, kurz ZXING [ZXIN] verwendet. Ursprünglich wurde die Bibliothek für das Scannen und Analysen von Barcodes entwickelt und um weitere Formate, wie den QRCode erweitert.

```
/**
 * Starts scanning process on BarcodeView
 */
public void startScan() {
    barcodeView.decodeSingle(new BarcodeCallback() {
        @Override
        public void barcodeResult(final BarcodeResult result) {
            String qrCode = result.getText();

            if (qrCode == null) {
                ((TransactionActivity) getActivity()).showToast(parentActivity
                        .getString(R.string.qr_code_unscanable));
                delayRescan();
            } else {
                // Add preview of scanned QRCode
                resultPreview.setImageBitmap(result.getBitmap());

                String[] qrCodeParts = qrCode.split(":");
                if (qrCodeParts.length < 2) {
                    ((TransactionActivity) getActivity()).showToast(parentActivity
                            .getString(R.string.qr_code_invalid));
                    delayRescan();
                } else {
                    String itemIdentifier = qrCodeParts[1];
                    parentActivity.addItem(itemIdentifier);
                }
            }
        }

        @Override
        public void possibleResultPoints(List<ResultPoint> resultPoints) {}
    });
}
```

Abbildung 11 - Scan eines QRCodes

Der Scan-Prozess wird mit Hilfe der Methode decodeSingle() gestartet. Sobald ein QRCode identifiziert und ausgelesen wurde, wird die Methode des Callback Objekts initiiert, welche prüft, ob es sich um einen Identifier handeln kann. Bei einem gültigen Identifier wird der eigentliche Aufruf initiiert.

b) Aufruf der API

Der Aufruf an die API wird innerhalb der Android Applikation mit der Bibliothek Retrofit [RET1] implementiert. Mit ihr ist es Entwicklern möglich, eine HTTP-API Innerhalb der Android Applikation wie ein Java Interface zu behandeln.

```
package com.xing.mobilerentalremoteapp.API;

import ...

public interface RentalToolAPI {
    // [...]

    // http://test.host/api/v1/add_item.json?transaction_id=1&identifier=identifier=AC000002
    @POST("/api/v1/add_item.json")
    Call<Response> addItem(@Query("transaction_id") Integer transaction_id,
                           @Query("identifier") String item_identifier);

    // [...]
}
```

Abbildung 12 - Interface für den API Aufruf addItem

Am Beispiel der abstrakten Methode addItem des API Interfaces wird mit der Annotation und unter Angabe der geforderten Parameter die Form der URL für den Aufruf vorgegeben. Durch die Übergabe des Interfaces an Retrofit wird ein Adapter für die API Schnittstelle zur Rails Anwendung erzeugt, welche alle Aufrufe in Form von Methoden innerhalb der Android Applikation zur Verfügung stellt. Beim Aufruf der addItem Methode mit den geforderten Parametern erzeugt der Adapter die URL für die Anfrage und sendet sie mit Hilfe des in Retrofit integrierten HTTP Clients an die Rails API.

c) Dispatcher und Routing

Innerhalb der Rails Anwendung trennt der Dispatcher beim Analysieren der URL des Aufrufs den Pfad auf der einen Seite und die Parameter transaction_id und identifier des Items auf der anderen. Anhand der Routingtabelle prüft er, ob es zu dem angegebenen Pfad eine verarbeitende Action innerhalb des Systems gibt.

```
72    # Rental-App API Routing
73    namespace :api, constraints: { format: :json } do
74      namespace :v1 do
75        post  'initiate'                    =>      'base#initiate'
76        get   'validate'                    =>      'base#validate'
77        post  'update'                      =>      'base#update'
78        post  'add_item'                    =>      'base#add_item'
79        post  'remove_item'                 =>      'base#remove_item'
80        get   'check_items'                 =>      'base#check_items'
81        post  'commit'                      =>      'base#commit'
82        post  'abort'                       =>      'base#abort'
83        post  'return_item'                 =>      'base#return_item'
84      end
85    end
```

Abbildung 13 - Routing Tabelle - Abschnitt API Schnittstelle

Innerhalb der Routing Tabelle sind die möglichen HTTP Anfragen an die Rails Anwendung definiert. Die API Schnittstelle der Rails Anwendung erlaubt nur Anfragen, welche explizit angeben, dass eine in JSON kodierte Antworten erwartet wird. Die API Schnittstelle ist durch den namespace */api/* vom Rest des Systems getrennt. Hierdurch wird sichergestellt, dass es nicht zu Überschneidungen bei den Routen für den APIController und den Routen anderer ActionController gibt. Auf dieselbe Weise werden weitere Versionen der selben API Schnittstelle voneinander getrennt, die im Laufe der Zeit für die Erweiterung des Systems noch entwickelt werden könnten. Anhand der Tabelle wählt der Dispatcher den APIController, instanziiert ihn und delegiert den Aufruf mit Übergabe der Parameter an die Action des Controllers.

d) Bearbeitung der Anfrage durch den APIController

Innerhalb der Action Methode stehen die Parameter des Aufrufs durch die Variable params zur Verfügung. Um den APIController möglichst frei von redundanter Funktionalität zu halten, ist die Validierung und Instanziierung der Objekte, die durch die Parameter referenziert werden, nicht innerhalb der Action implementiert, sondern in eine eigene Methoden ausgelagert.

```ruby
module Api
    module V1
        class BaseController < ApplicationController
            before_action :set_transaction, only: [:validate, :item_status, :add_item, :remove_item,
                                                   :update, :check_items, :commit, :abort]
            # [...]
            private
            # Sets instance variable transaction from params
            def set_transaction
                render_error([{
                    status: :transaction_id_not_specified,
                    text: "Transaction ID not specified"
                }]) and return if params[:transaction_id].nil?
                @transaction = ApiTransaction.find_by_id(params[:transaction_id])
                render_error([{
                    status: :transaction_not_found,
                    text: "Transaction with ID '#{params[:transaction_id]}' not found"
                }]) if @transaction.nil?
            end
            # [...]
        end
    end
end
```

Abbildung 14 - APIController before_action

In diesem Beispiel wird die Transaction innerhalb der set_transaction Methode validiert, als ActiveRecord Objekt instanziiert und an eine instanzweite Variable gebunden. Auf diesem Weg steht sie nach Ausführung der Methode allen folgenden Operationen innerhalb des ActionControllers zur Verfügung. Für Actions die eine gültige Transaction benötigen wird hier mit Hilfe eines sogenannten before_filters festgelegt, dass im Vorfeld die set_transaction Methode ausgeführt werden soll. Wenn bei der Anfrage an die Rails Anwendung keine gültige Transaction referenziert wird, so bricht die Methode durch Aufruf von render_error mit der Generierung einer Antwort vorzeitig ab.

```ruby
# Adds an item to the transaction
def add_item
    if @transaction.add(@item)
        render_success({
            status: :item_added,
            text: "Item has been added to your #{@transaction.transaction_type.titleize}"}
        )
    else
        render_error([{
            status: :item_already_part_of_transaction,
            text: "This item already is part of your #{@transaction.transaction_type.titleize}"
        }])
    end
end
```

Abbildung 15 - APIController - add_item

Innerhalb der add_item Methode sind Transaction und, analog dazu, Item nach Ausführeng der before_filter als Variablen verfügbar. Abhängig davon, ob das

Hinzufügen des Items zur Transaction gelingt oder nicht, generiert der ActionController als letzten Schritt seine Antwort in der JSON Notation und sendet sie an die Android Applikation. Diese Antwort beinhaltet unter anderem die Transaktion in serialisierter Form.

e) Verarbeitung der Antwort in der Android Applikation

Bei dem ursprünglichen Aufruf der API Schnittstelle wird ein Callback Objekt erzeugt und an den Aufruf gebunden. Der Callback ist ein selbständig lauffähiger Java Thread, der asynchron zur Ausführung der Android Applikation auf die Antwort der API Schnittstelle wartet und diese beim Eintreffen bearbeitet.

```
/**
 * Adds an Item to the Transaction, based on the Item's Identifier
 * @param itemIdentifier Item's Identifier
 */
public void addItem(String itemIdentifier) {
    progressBar.setVisibility(View.VISIBLE);
    APIFactory.getInstance().addItem(getTransactionID(), itemIdentifier)
            .enqueue(new Callback<Response>() {
        @Override
        public void onResponse(Call<Response> call, retrofit2.Response<Response> apiResponse) {
            progressBar.setVisibility(View.INVISIBLE);
            Response response = apiResponse.body();
            if (response.success) {
                Log.d(TAG, "addItem().onResponse() with success");
                transaction = response.transaction;
                getOverviewFragment().updateOverview(transaction);
                pagerAdapter.setNumberOfItems(transaction.items.size());
                if (Config.GO_TO_OVERVIEW_AFTER_SCAN) {
                    openOverview(null);
                }
                showToast(response.message.text);
                getScannerFragment().delayRescan();
            } else {
                // [...]
            }
        }

        @Override
        public void onFailure(Call<Response> call, Throwable t) {
            // [...]
        }
    });
}
```

Abbildung 16 - Callback des API Aufrufs

An dieser Stelle wird ein weiterer Aspekt der Retrofit Bibliothek sichtbar. Die Objekte, die in serialisierter Form in der Antwort der API vorliegen werden mit Hilfe einer sogenannten CallAdapter Factory[RET2] zu vollwertigen Java Objekten deserialisiert. Damit dies möglich ist, wurden im Vorfeld bei der Entwicklung der Android Applikation Modelle dieser Objekte angelegt. Bei der Deserialisierung und Instanziierung der Objekte belegt Retrofit die Attribute entsprechend der Antwort der API. Wie durch die Abbildung gezeigt wird, ist es hiermit möglich, bei der Bearbeitung der Antwort auf die Attribute der Objekte Response, Transaction und Message zuzugreifen. Hiermit kann die Android Applikation auswerten, ob der Aufruf erfolgreich war und anhand weiterer Interaktionen durch den Benutzer mit der Transaction fortführen.

5.3 Optimierung der Reaktionszeit des Systems

Wie im Kapitel Grundlagen beschrieben, ist es möglich die Zugriffszeiten auf die Weboberfläche des Systems, durch partielles Laden von Inhalten stark zu verbessern. Am Beispiel der Anzeige von Detailinformationen von Items wird in diesem Abschnitt das Vorgehen bei der Implementierung eines partiellen Ladens von Inhalten gezeigt. Die Umsetzung setzt sich aus zwei Teilaspekten zusammen. Zum einen der Aufruf im Browsers des Benutzers mit Hilfe von JavaScript und zum anderen eine Interaktionsschnittstelle der Rails Anwendung, über welche die Anfrage des Browsers auf der Serverseite bearbeitet wird.

a) Initialisierung des partiellen Ladens durch Javascript

```
function initModalLoader() {
  // Unbind dynamic modal loading event if already present(turbolinks)
  $(document).off('click', 'a[data-modal]');

  // Bind dynamic modal loading event
  $(document).on('click', 'a[data-modal]', function() {
    var url = $(this).attr('href');
    var target = $(this).attr('data-target');

    $.get(url, function(data) {
      $(target).html(data).find('.modal').modal();
    });
    return false;
  });
}
```

Abbildung 17 - JavaScript Function - ModalLoader

Mit Hilfe der on() Funktion wird innerhalb eines HTML Dokuments das Standardverhalten aller Hyperlinks einer bestimmten Struktur überschrieben. Besitzt ein Hyperlink das Attribute data-modal, so wird statt eines normalen Seitenaufrufs die angegebene JavaScript Funktion aufgerufen. Neben der Adresse erwartet die Funktion innerhalb des Hyperlink-Elements eine Referenz auf ein Zielelement. In dieses Zielelement wird der geladene Inhalt innerhalb der HTML Baumstruktur eingehängt.

```
<!DOCTYPE html>
<html lang="en">
  [...]
  <body>
    [...]
    <div id="modal-placeholder"></div>
    [...]
    <a data-modal="true" data-target="#modal-placeholder"
     class="btn-link" href="/devices/2/details_modal">Plus One</a>
    [...]
  </body>
</html>
```

Abbildung 18 - Vereinfachte HTML Dokument mit Modal Link

Anhand der Abbildung wird gezeigt, wie ein solcher Link aufgebaut ist. Das Zielelement kann dabei an einer beliebigen Stelle innerhalb des HTML Dokumentes stehen.
Bei einem Klicken dieses Links wird die JavaScript Funktion aufgerufen. Sie lädt mit Hilfe der Adresse eine HTML Teilstruktur vom Server und bindet sie innerhalb des Ziels in das HTML Dokument ein. Diese Struktur beinhaltet die Darstellung der

detaillierten Informationen eines Gerätes in HTML. Mit einem Aufruf der modal() Funktion im Anschluss an das Laden wird diese Struktur für den Benutzer sichtbar gemacht.
Um den Aufruf möglich zu machen, muss innerhalb der Rails Anwendung der Aufruf der JavaScript Funktion bearbeitet werden.

b) Bearbeitung des Aufrufs durch den ItemController

```ruby
# GET /items/1/details_modal
def details_modal
  rentals = @item.rented
  reservations = @item.reserved

  # Evaluate item status based on priority
  @status = if @item.marked_as_deleted
    :deleted
  elsif rentals.any?
    :rented
  elsif reservations.any?
    :reserved
  else
    :free
  end

  @blocking = { rentals: rentals, reservations: reservations }

  @html_fallback = request.format == Mime::HTML

  respond_to do |format|
    format.html { render(layout: @html_fallback) }
  end
end
```

Abbildung 19 - Modal Aufruf aus sicht des ActionControllers

Die Detailinformationen des Items werden innerhalb der details_modal Methode erzeugt und mit einem Aufruf der render() Methode an das HTML Templates der Methode gereicht. Dieses generiert die HTML Struktur für die Darstellung der Informationen, welche als Antwort an den Browser gesendet wird.
Für den Fall, dass auf einem Browser kein JavaScript ausgeführt werden kann, besitzt die Methode eine Ausweichstrategie um dennoch die Anzeige der Detailinformationen auf dem Browser zu ermöglichen. Ein solcher Browser interpretiert den Hyperlink der in Abbildung 19 zu sehen ist als ganz normalen Seitenaufruf. Dieser Seitenaufruf kann innerhalb der Methode des ItemController

anhand des Formats des Aufrufs identifiziert werden. Handelt es sich also um einen HTML Aufruf, so wird das HTML Template für die Antwort in das Layout der Rails Anwendung gebettet und erzeugt damit statt einem HTML Teilstruktur ein vollständiges gültiges HTML Dokument. Hierdurch ist es dem Browser möglich einen Seitenwechsel das erhaltene HTML Dokument zu vollziehen.

6 Zusammenfassung

Es war eine sehr spannende Erfahrung, dieses vergleichsweise komplexe System von den Anfängen in den Spezifikationsmeetings, über den Entwurf und die testgetriebene Entwicklung, bis hin zum Feinschliff in Sachen Optimierung der Leistung und der Benutzerfreundlichkeit in jedem Detail zu verwirklichen. Hierfür standen mir hoch spezialisierte Kollegen aus vielen verschiedenen Bereichen der Software Entwicklung zur Seite, die immer dazu bereit waren das Für und Wider von Konzepten und Strategien in Hinblick auf die Realisierung des Projektes zu diskutieren.
Die gesetzten Ziele wurden erfüllt und das System ist von den Mitarbeitern des Unternehmens angenommen worden. Hierbei wurde die Bedienung als sehr intuitiv wahrgenommen und die Verbesserung des Prozesses für das Ausleihen und Reservieren von Betriebsmitteln mit dem neuen System gelobt.

6.1 Ausblick

Es bietet sich an, dass System um die Funktion zur Verlängerung der Ausleihfrist von Betriebsmitteln zu erweitern. Obwohl diese Funktion sehr naheliegend ist, ist sie nicht im Zuge der Spezifikationsmeetings angefragt worden, sondern erst sehr spät während das System bereits fertig gestellt war. Leider hat es am Ende zeitlich nicht mehr gereicht, diese Funktion umzusetzen.
Darüber hinaus ist es denkbar, dass das System in Zukunft in weiteren Bereichen des Unternehmens zum Einsatz kommt. Das Team Internal IT zum Beispiel verwaltet die permanente Ausgabe von Betriebsmitteln an alle Mitarbeiter. Hier könnte eine eigenständige Version des Systems, mit auf den Bedarf der Internal IT abgestimmter Funktionalität zum Einsatz kommen.

Quellen

[SPIL10] SPILLNER, Andreas; LINZ Tilo; Basiswissen Softwaretest – Aus- und Weiterbildung zum Certified Tester; 2010; dpunkt Verlag

[CHEL10] CHELIMSKY, David u. A.; The RSpec Book – Behaviour-Driven Development with RSpec, Cucumber, and Friends; 2010; The Pragmatic Bookshelf

[BECK15] BECKER, Arne; PANT, Marcus; Android 5 – Programmieren für Smartphones und Tablets; 2015; dpunkt Verlag; S. 26-33

[ASYN] Async Attribute
https://www.w3.org/TR/html/scripting-1.html#attr-script-async
Letzter Aufruf 24.04.16

[FACA] Facebook API Übersicht
https://developers.facebook.com/
Letzter Aufruf 20.04.16

[GITA] GitHub API Übersicht
https://developer.github.com/v3/
Letzter Aufruf 20.04.16

[GOOA] Google API Übersicht
https://developers.google.com/apis-explorer/
Letzter Aufruf 20.04.16

[HT11] Methods of HTTP/1.1
https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
Letzter Aufruf 18.04.16

[HTM5] HTML 5
https://www.w3.org/TR/html5/

Letzter Aufruf 23.04.16

[JSON] JSON
http://www.json.org/json-de.html
Letzter Aufruf 27.04.16

[RET1] RetroFit
http://square.github.io/retrofit/
Letzter Aufruf 03.05.16

[RET2] Retrofit CallAdapter Factory
https://square.github.io/retrofit/2.x/retrofit/retrofit2/CallAdapter.Factory.html
Letzter Aufruf 03.05.16

[RSPE] RSpec
https://github.com/rspec/rspec
Letzter Aufruf 29.04.15

[ROR1] Einstieg Rails
http://guides.rubyonrails.org/index.html
Letzter Aufruf 12.04.16

[ROR2] ActiveRecord API
http://api.rubyonrails.org/classes/ActiveRecord/Base.html
Letzter Aufruf 12.04.16

[ROR3] ActiveRecord Inheritance
http://api.rubyonrails.org/classes/ActiveRecord/Inheritance.html
Letzter Aufruf 15.04.16

[ROR4] ActionController API
http://api.rubyonrails.org/classes/ActionController/Base.html
Letzter Aufruf 18.04.16

[ROR5] Routing API
http://api.rubyonrails.org/classes/ActionDispatch/Routing.html
Letzter Aufruf 18.04.16

[ROR6] ActionView API
http://api.rubyonrails.org/classes/ActionView/Base.html
Letzter Aufruf 14.04.16

[ROR7] ActionMailer API
http://api.rubyonrails.org/classes/ActionMailer/Base.html
Letzter Aufruf 21.04.16

[ROR8] ActiveJob API
http://api.rubyonrails.org/classes/ActiveJob/Base.html
Letzter Aufruf 21.04.16

[ROR9] QueueAdapters
http://edgeapi.rubyonrails.org/classes/ActiveJob/QueueAdapters.html
Letzter Aufruf 21.04.16

[ROR10] Namespaces
http://guides.rubyonrails.org/routing.html#controller-namespaces-and-routing
Letzter Aufruf 18.04.16

[TURB] Turbolinks
https://github.com/turbolinks/turbolinks
Letzter Aufruf 24.04.16

[TWIA] Twitter API Übersicht
https://dev.twitter.com/rest/public
Letzter Aufruf 20.04.16

[W3DO] Document Object Model
https://www.w3.org/DOM/
Letzter Aufruf 24.04.16

[ZXIN] Zebra Crossing
https://github.com/zxing/zxing
Letzter Aufruf 02.05.16

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____