Fachbereich Informatik und Medien

BACHELORARBEIT

Entwurf und prototypische Implementierung einer Android-Anwendung zur Bestandsverwaltung und Dokumentation in einem Rechenzentrum

vorgelegt von:

Jens Seidemann
geboren am: 07.04.1986

zum
Erlangen des akademischen Grades

BACHELOR OF SCIENCE
(B.Sc.)

Erstbetreuer: Prof. Dr.-Ing. Thomas Preuß
Zweitbetreuer: Dr.-Ing. Thomas Streubel

Danksagung

Auf diese Weise möchte ich mich bei meinen Betreuern Prof. Dr. Thomas Preuss und Dr. Thomas Streubel für die Unterstützung bei der Erstellung dieser Arbeit bedanken.

Weiterhin gilt mein besonderer Dank Marco Felten, der mir sein Vertrauen schenkt und mich seit Jahren unentwegt fördert.

Zudem gilt mein Dank Stephan Klasen, der mich seit Jahren fördert, fordert und mich immer wieder zu neuen Ideen anregt.

Nicht zu vergessen, möchte ich meinen Eltern danken, ohne die die Mission Studium und alle dazugehörigen Probleme nicht zu bewältigen gewesen wären.

Abschließend möchte ich mich bei meiner Freundin bedanken, die immer ein offenes Ohr für mich hat und stets meinen Launen standhält.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit zum Thema

„Entwurf und prototypische Implementierung einer Android-Anwendung zur Bestandsverwaltung und Dokumentation in einem Rechenzentrum“

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Brandenburg an der Havel, 22.08.2011

Jens Seidemann

Kurzfassung

Die Verwaltung moderner Rechenzentren gestaltet sich oft sehr umfangreich. Neben dem technischen Betrieb der Komponenten ist auch die Dokumentation und die Erfassung von Inventardaten ein wichtiger Bestandteil der Arbeiten. Die Web-Anwendung „Data Center Asset Management Solutions“ der Firma „speedikon®“ wird in der Verlag Der Tagesspiegel GmbH unterstützend für diese Aufgaben eingesetzt. In dieser Arbeit wird eine prototypische Anwendung auf Basis des Betriebssystems Android realisiert, die die vorhandenen Inventardaten der Web-Anwendung für mobile Geräte verfügbar macht. Der Zugriff auf die Datenbank wird durch einen Web-Service realisiert, dessen Planung und Umsetzung beschrieben wird. Ergänzend werden am Beispiel der mobilen Anwendung beispielhaft Unit-Tests implementiert, um Möglichkeiten für das Testen von Android-Anwendungen aufzuzeigen. Für die Themen Web-Service und Unit-Tests in Android erfolgt zuvor eine theoretische Betrachtung. Als Grundlage für die durchgeführten Implementierungen werden die angeboten Funktionen und die Datenbank von „speedikon® DAMS“ erläutert.

Abstract

Management of modern data centers is often very extensive. In addition to technical operation of components the documentation and collection of inventory data is also an important part of the work. The web application "Data Center Asset Management Solutions" developed by the company "speedikon®" is used in the publishing company “Der Tagesspiegel”, to support these tasks. In this thesis a prototype application based on Android operating system is realized, which makes the existing inventory data available for mobile devices. Access to the database is realized through a web service, whose planning and implementation is described. In addition unit tests are implemented exemplarily, to identify potentials for testing of Android applications. For web service and unit testing in Android a theoretical preview has been written. As a basis for the implementations the functions offered by “speedikon® DAMS“ and the database are analyzed.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die Verwaltung zweier Rechenzentren in einem Zeitungsverlag mit täglicher Produktion, wie bei der Verlag Der Tagesspiegel GmbH, ist sehr komplex, da dafür die unterschiedlichsten Geräte- und Inventardaten der verbauten Komponenten erfasst werden müssen. Naturgemäß ist dabei auch die Fehlersuche während der laufenden Produktion besonders zeitkritisch, d.h. die fehlerhaften Geräte müssen schnell lokalisiert werden können. Um diese Aufgaben zu bewältigen, wurde die Web-Anwendung „speedikon® DAMS“ (Data Center Asset Management Solutions) beschafft. Diese Softwarelösung unterstützt die Verwaltung und Visualisierung von Inventardaten in den Rechenzentren. Die Pflege dieser Daten erfolgt im Moment ausschließlich über eine Weboberfläche, die lediglich einen PC-Arbeitsplatz mit entsprechendem Web-Browser voraussetzt. Die Herstellerfirma „speedikon®“ bietet zudem eine portable Lösung auf Grundlage von PDAs an. Der Nachteil in dieser Lösung besteht darin, dass die Daten lokal auf dem PDA vorgehalten werden und somit eine ständige Synchronisierung zwischen der Datenbank und dem PDA erforderlich ist.

Die Idee in der vorliegenden Arbeit ist es, die steigende Nutzung von mobilen Endgeräten, wie Apple iPad® oder Samsung® Galaxy Tab, auch in den administrativen Bereich des Verlages einzubinden und somit die Anschaffung von gesonderten PDAs zu vermeiden. So soll mit Hilfe der mobilen Endgeräte die Nutzbarkeit der bereits aufgenommen Daten flexibler gestaltet werden, um mit kurzen Zugriffszeiten die aktuellen technischen Daten, Verkabelungswege und Standorte der einzelnen Komponenten wiedergeben zu können. Neben dem Echtzeitzugriff der Daten können durch die Nutzung portabler Endgeräte auch weitere Vorteile genutzt werden. Die einfache Bedienbarkeit, die geringen Abmaße der Geräte und die in den meisten Fällen integrierte Kamera in Verbindung mit einer portablen Anwendung, können unter Umständen zu einer deutlichen Akzeptanzsteigerung der genannten Web-Anwendung führen.

1.2 Aufgabenstellung und Abgrenzung

In diesem Projekt soll der Prototyp einer Android-Anwendung entwickelt und implementiert werden, der die vorhandenen Daten von „speedikon® DAMS“ nutzt. Die mobile Anwendung soll eine grafische Benutzeroberfläche zur Verfügung stellen, die nach einer erfolgreichen Anmeldung des Benutzers die Suche und Abfrage von Inventardaten der Rechenzentren mit Hilfe diverser Suchkriterien ermöglicht und eine entsprechende Ausgabe generiert. Mit Hilfe der entstehenden Anwendung soll zudem verifiziert werden, ob der Einsatz von mobilen Endgeräten im administrativen Bereich sinnvoll ist und zur erwarteten Akzeptanz sowie zur Verbesserung der Verwaltungs-Arbeiten in den Rechenzentren führt.

Für das Projekt gelten diverse Einschränkungen. Während der Entwicklung der Anwendung wird ausschließlich eine lokale Kopie der vorhandenen Datenbank genutzt. Grund dafür ist die aus Sicherheitsgründen fehlende Anbindung der mobilen Endgeräte über WLAN an das vorhandene Server-Netzwerk des Verlages (s. *3.2 Verwendung in der Projektumgebung*). Aus diesem Grund muss die Anbindung an die produktive Datenbank das Thema eines weiteren Projektes sein. In der Planung muss zudem berücksichtigt werden, dass die Anwendung beim produktiven Einsatz auf Endgeräten unterschiedlicher Hersteller lauffähig sein soll. Die im Rahmen dieser Arbeit aufgezeigte Umsetzung in Android stellt somit nur eine erste Test-Plattform dar.

Ergänzend soll auf die Grundlagen zum Testen von Android-Anwendungen eingegangen werden und am Beispiel einiger Test-Klassen erläutert werden. Aufgrund der Komplexität möglicher Testszenarien soll dabei das Hauptaugenmerk auf das Testen android-spezifischer Komponenten gelegt werden.

1.3 Aufbau und Methodik

Die vorliegende Arbeit ist in fünf Teilabschnitte gegliedert. Nach einer Einführung mit der Beschreibung der Motivation und der Aufgabenstellung folgt in Kapitel 2, die theoretische Vorbetrachtung des Projektes. Dabei wird auf die theoretischen Grundlagen eines selbst zu implementierenden Web-Service eingegangen. In Kapitel 3 erfolgt die Beschreibung der Web-Anwendung „speedikon® DAMS“ und ein kurzer Überblick zum Thema Android Betriebssystem. Als Abschluss der theoretischen Betrachtung wird eine kurze Einführung in diverse Softwaretests gegeben. Diese Einführung soll als Grundlage für die weitere Betrachtung von

Unit-Tests dienen, die dann als Tests für die Android Umgebung im weiteren Verlauf der Arbeit umgesetzt werden.

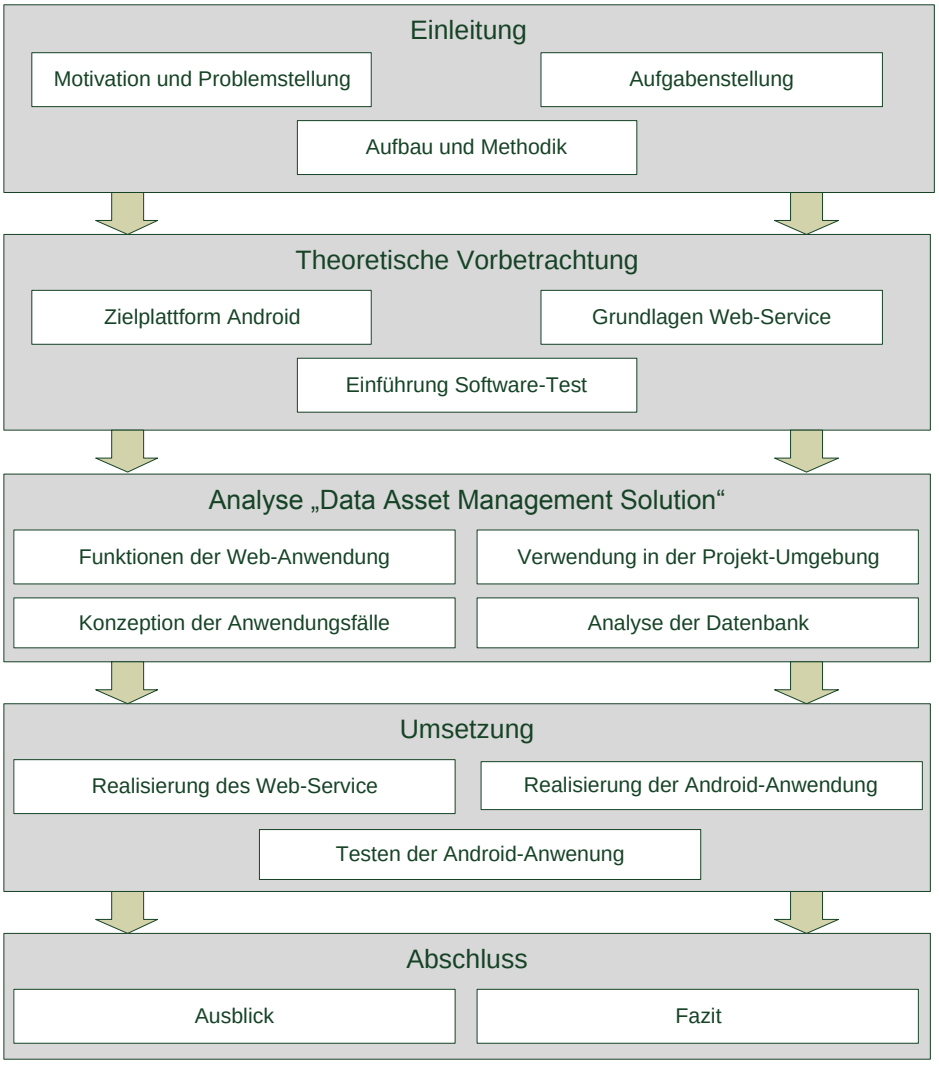

Abbildung 1: Aufbau der Arbeit

Im Anschluss folgen im vierten Kapitel die Abschnitte zur Umsetzung des Web-Services, der mobilen Anwendung und der Realisierung der Unit-Tests. Im Kapitel 5 wird ein Ausblick auf mögliche Erweiterungen der implementierten Anwendung gegeben. Den Abschluss der Arbeit bildet ein Fazit, dass die Ergebnisse der Betrachtungen kurz zusammengefasst. *Abbildung 1* fasst den Aufbau der Arbeit nochmals grafisch zusammen.

2 Theoretische Vorbetrachtung

2.1 Web-Service

Durch die Einschränkung, dass durch die mobilen Endgeräte kein direkter Zugriff auf das interne Server-Netzwerk erfolgen darf, muss eine Möglichkeit gefunden werden, einen abgesetzten Zugriff auf die Daten der Web-Anwendung zu erstellen. Ein möglicher Ansatz stellt dabei die Implementierung eines Web-Service dar.

Web Services spielen im Zeitalter des Internets eine wichtige Rolle. Die Idee bei der Realisierung der Web-Services ist die Bereitstellung vorhandener Anwendungen zur Verwendung im Internet bzw. über ein internes Netzwerk. Bekannte Web-Anwendungen wie Amazon, Ebay und Google bieten Schnittstellen zur Nutzung der eigenen Dienste für die Einbettung in anderen Anwendungen an.

Für den Austausch und die Manipulation von Daten über definierte Schnittstellen hat sich die Architektur Representational State Transfer (REST) etabliert (vgl. [Sch11]), für die Roy Fielding im Jahr 2000 in seiner Doktorarbeit (s. [Fie00]) die Grundlagen definiert hat.

Die zentrale Einheit bei der REST-Architektur, stellen die so genannten Ressourcen dar. Ressourcen sind in diesem Fall Inhalte, die über das Netzwerk aufgerufen werden können. Diese Inhalte können in Form von Dokumenten, Bildern oder auch als Ergebnisse von Datenbankabfragen vorliegen, die in unterschiedlichen Formaten wie XML oder JSON zur Verfügung gestellt werden. JSON stellt dabei ein plattformunabhängiges Datei-Austauschformat dar. Im Gegensatz zu XML, verzichtet JSON auf die Verwendung von Tags zur Darstellung der Daten und produziert somit weniger Overhead. Ein ausführlicher Vergleich ist in [Jso11] zu finden.

Für den Aufruf wird jede der Ressourcen mit einem eindeutigen „Uniform Resource Identifier“ (URI) adressiert. Die Idee bei REST liegt in der einfachen Manipulation der angesprochen Ressourcen, wobei über einfache Schnittstellen die so genannten CRUD-Methoden durchführbar sein sollen.

Die Bezeichnung CRUD steht dabei für folgendes:

- Create
    - Erzeugen einer neuen Ressource
- Read
    - Lesen/ Aufruf einer Ressource
- Update
    - Änderungen an einer Ressource
- Delete
    - Löschen einer Ressource

Als Transportprotokoll nutzt REST das im Internet weit verbreitete Hyper Text Transfer Protocol (HTTP). Die Nutzung von HTTP macht die Web-Anwendung somit plattformunabhägig, da für die Nutzung der Schnittstelle lediglich ein HTTP-Client benötigt wird, der in vielen Programmiersprachen bereits zur Verfügung steht. Neben der geringen Anzahl an Methoden (s. [Wor11]) bietet HTTP noch weitere Vorteile. Die Methode GET als lesende Operation kann durch den Client zwischengespeichert werden, so dass bei einem erneuten Aufruf der Ressource die Wartezeit und die Netzwerklast verringert werden kann. Weiterhin bietet HTTP aus Sicht der Netzwerksicherheit den Vorteil, dass der Port von HTTP (TCP/UDP 80) durch die meisten Firewall-Systeme nicht blockiert wird, da dieser Port die Grundlagen alltäglicher Internetzugriffe darstellt.

Eine weitere grundlegende Eigenschaft der REST-Architektur ist die zustandslose Kommunikation. Zustandslos bedeutet hier, dass der Server keine Sitzungsdaten der Nutzeranfragen vorhält. Wenn die Verwaltung der Sitzungsdaten für eine Anwendung erforderlich ist, wird dies durch das dienstanfragende Gerät realisiert. Die Sitzungsdaten werden dann in jeder HTTP-Anfrage mitgesendet und entsprechend durch den Server ausgewertet.

In der Programmiersprache Java™ steht seit 2008 die „Java™ API for RESTful Web-Services“ (JAX-RS) zur Verfügung und stellt Funktionalitäten bereit, um „RESTful“-Web-Services zu implementieren. Eine Referenz-Implementierung für JAX-RS stellt das Framework „Jersey[1]“ dar, das unteranderem die Möglichkeit zur Einbindung von Annotationen bietet, um Meta-Daten, die für den Web-Service benötigt werden, einzubinden. (s. [Bur10])

[1] http://jersey.java.net

2.2 Zielplattform Android

Für die exemplarische Umsetzung der Projektaufgabe wurde die Verwendung von Android als Betriebssystem vorgesehen. Die erste Entwicklung von Android erfolgte durch die gleichnamige Firma, bevor diese durch Google aufgekauft wurde. Android erfuhr seit der Einführung 2007 als Handy-Betriebssystem diverse Weiterentwicklungen. So erfolgte am Anfang des Jahres 2011 die Erweiterung auf Version 3.0, die speziell für Tablet-PCs angepasst wurde (s. [Jun11]). Für die Android-Umgebung steht ein Software-Development-Kit[2] als Plug-In für die Entwicklungsumgebung „Eclipse[3]“ zur Verfügung, mit dem es möglich ist, eigene Anwendung zu implementieren. Weiterhin gibt es daneben noch quelloffene Implementierungen, die in eigene Anwendungen eingebunden werden können. Dazu zählt unter anderem die Barcode-Anwendung „Zxing[4]“, die es mit Hilfe der eingebauten Kamera ermöglicht, Strichcodes (z.B. EAN-Code, ISBN) einzuscannen.

Grundlage für die vorliegende Arbeit stellt die Android-Version 2.2 auf dem Samsung® Galaxy Tab dar.

2.3 Software-Tests

2.3.1 Einführung

Das Testen ist ein wesentlicher Bestandteil in der Softwareentwicklung, um eine hohe Qualität des erstellten Produktes zu gewährleisten. Die Entwicklung von Testszenarien und den entsprechenden Funktionstests sollte so früh wie möglich erfolgen, so dass auftretende Fehler möglichst wenige Auswirkungen auf spätere Entwicklungen haben können. Nachfolgend sollen kurz mögliche Typen von Softwaretests nach [HT03] aufgeführt werden:

- Unit-Test

  Die Unit-Tests sind Testklassen, in denen Methoden implementiert werden, um einzelne Programm-Module isoliert, d.h. ohne Einwirkung von anderen Modulen, zu testen. Unit-Tests stellen somit die Grundlage für weitere Tests dar.

[2] http://www.developer.android.com/sdk/index.html
[3] http://www.eclipse.org
[4] http://code.google.com/p/zxing

- Integrationstest

  Diese Tests stellen die nächste Stufe der Softwaretests nach den Unit-Tests dar. Im Gegensatz zu den Unit-Tests, werden die Abhängigkeiten und Funktionsweisen von Modulen untereinander betrachtet.

- Validierung und Verifikation

  Aufgabe dieser Tests ist es, herauszufinden, ob die erstellte Anwendung den gestellten Ansprüchen des Auftraggebers entspricht. Zudem muss geklärt werden, ob die Anforderungen des Auftraggebers auch den Wünschen der Anwender gerecht werden.

- Ressourcenverbrauch

  Neben der korrekten Funktionsweise des erstellten Quelltextes ist auch die Lauffähigkeit auf dem Zielsystemen zu überprüfen. Die verfügbaren Ressourcen des Zielsystems sind zudem begrenzt. Dazu zählen unteranderem die Leistungsfähigkeit der CPU, sowie der Speicher- und Festplattenplatz. Zudem kann auch die Darstellung zwischen einzelnen Endgeräten variieren.

- Performance-Tests

  Diese Tests sollen die Leistungsfähigkeit und die Leistungsgrenzen einer Anwendung aufzeigen. Dabei muss hinterfragt werden, ob die gewünschte Anzahl der Benutzer und Verbindungen unterstützt werden kann. Diese Betrachtung ist vor allem für die Skalierbarkeit der Anwendung wichtig.

- Usability-Tests

  Aus Sicht der Anwender stellen Usability-Tests einen sehr wichtigen Aspekt dar. Bei diesen Tests stellt sich heraus, ob die erstellte Anwendung den Benutzer-Anforderungen entspricht und ob die gewünschten Funktionen abgedeckt sind. Zudem soll durch die Endanwender die Benutzerbarkeit getestet und eingeschätzt werden. Somit liefern Usability-Tests den Entwicklern wichtige Erkenntnisse zur schrittweisen Verbesserung der Benutzerfreundlichkeit.

Das Testen aller aufgezeigten Möglichkeiten gestaltet sich sehr umfangreich. Aus diesem Grund soll in den folgenden Abschnitten ausschließlich auf die Grundla-

gen von Unit-Tests und deren Implementierung für eine Android-Anwendung eingegangen werden. Die Grundlagen bildet das JUnit-Framework, das näher beschrieben wird. Weiterhin werden Test-Möglichkeiten aufgezeigt, die an Android-Anwendungen angepasst sind.

2.3.2 JUnit-Framework

Das JUnit-Framework ist aus dem von Kent Beck entwickelten SUnit-Framework für die Programmiersprache Smalltalk hervorgegangen (s. [RAI05]) und stellt Methoden zur Verfügung um Unit-Tests durchzuführen. Unter anderem zählen dazu Methoden, mit denen die Testumgebungen vorbereitet werden können, die die Ergebnisse visualisieren und die getesteten Objekte freigeben. Die zu implementierenden Test-Klassen stellen eine Unterklasse der Klasse *TestCase* aus dem JUnit-Framework dar, die die benötigten Methoden bereitstellt. Nachfolgend sollen einige Methoden aus der Klasse *TestCase* aufgeführt werden.

- *setUp()*

  Die Methode *setUp()* schafft die Umgebungsbedingungen für die aufrufenden Test-Methoden. Dabei werden die für den Unit-Test benötigten Objekte initialisiert. Die *setUp()* Methode wird vor jedem Aufruf einer Methode aus der erstellten Test-Klasse aufgerufen, um die Objekte für den nächsten Test erneut zu initialisieren, so dass für jede Test-Methode die gleichen Ausgangswerte vorliegen.

- *tearDown()*

  Die Methode *tearDown()* wird nach jedem Aufruf einer Test-Methode aufgerufen, um die zuvor für den Test erzeugten Objekte und gebundenen Ressourcen wieder freizugeben.

- *run()*

  Diese Methode führt die Test-Klasse aus und sammelt die Ergebnisse der einzelnen Test-Methoden ein.

Neben den Klassen zur Vor- und Nachbereitung der Testumgebung stehen in der Klasse *Assert* Methoden zur Verfügung, um Behauptungen aufzustellen und zu überprüfen, ob diese erfüllt werden. Im Folgenden sollen die entsprechenden Methoden kurz beschrieben werden.

- *assertEquals([String nachricht], erwartet, derzeitig)*

  Diese Methode stellt eine Möglichkeit zur Verfügung, um einen erwarteten Wert mit dem derzeitigen Wert zu vergleichen, den der zu testende Quellcode bereitstellt. Die optional anzugebende Nachricht wird bei einem negativen Testergebnis ausgegeben.

- *assertNull([String nachricht], objekt)*

  Die Methode liefert einen Fehler, wenn das zu testende Objekt nicht null ist. Zusätzlich gibt es auch die Methode assertNotNull, die einen Fehler generiert, wenn das Objekt null ist.

- *assertSame([String nachricht], erwartet, derzeitig)*

  Die Methode *assertSame()* bietet die Möglichkeit zu testen, ob es sich um das gleiche Objekt handelt. Der Test schlägt fehl, wenn das erwartete und das derzeitige Objekt nicht das gleiche Objekt darstellen. Die Umkehrung für diese Methode ist die *assertNotSame()*-Methode.

- *assertTrue([String nachricht], wahrheitswert)*

  Diese Methode erwartet, dass der gegebene Wahrheitswert wahr ist. Die Umkehrung dieser Methode ist die *assertFalse()*-Methode, die einen falschen Wahrheitswert als Behauptung erwartet.

- *fail([String nachricht])*

  Mit Hilfe der Methode *fail()* kann ein Test abgebrochen werden. Als Einsatzszenario für diese Methode kann unter anderem das Testen von Ausnahmesituationen (Exceptions) angesehen werden. Das folgende Code-Beispiel (s. *Listing 1*) soll die Funktionsweise kurz verdeutlichen.

```
public void testException() {
        try{
            methode4(null);
            fail("Die Methode sollte eine Exception werfen!");
        } catch (Exception e){
            assertTrue(true);
        }
}
```

Listing 1: Beispiel zur Verwendung der Methode *fail()*

Es wird davon ausgegangen, dass die aufgerufene Methode eine Exception wirft, wenn der Eingabeparameter *null* ist. Wird die Exception geworfen, wird die Methode *assertTrue(true)* im catch-Block aufgerufen und der Test somit als erfolgreich ausgegeben. Wird die Exception nicht erzeugt, wird die Methode *fail()* mit der entsprechenden Nachricht aufgerufen und somit erzwungen, dass der Test nicht erfolgreich ist.

2.3.3 Testen in Android

Im folgenden Abschnitt sollen die Möglichkeiten der Software-Tests von Android-Anwendungen nach [And11] näher beschrieben werden.

2.3.3.1 Grundlagen

Die in *2.3.3 JUnit Framework* beschriebenen Klassen stellen die Grundlage für das Testen von Android-Anwendungen dar. Im Folgenden soll kurz beschrieben werden, welche Funktionalitäten für das Testen der Anwendungen zur Verfügung stehen

- AndroidTestCase

  Für die zu implementierenden Testklassen steht die Klasse *AndroidTestCase* zur Verfügung, die die *TestCase* Klasse aus dem JUnit-Framework erweitert, so dass spezielle *setUp()* und *tearDown()*- Methoden für Android-Umgebungen zur Verfügung stehen.

- Assertation

  Die beschriebenen Assert-Methoden des JUnit-Frameworks können durch die Vererbung aus der *Assert*-Klasse genutzt werden. Für die speziellen Bedürfnisse einer Android-Anwendung sind weitere *Assert*-Methoden in den Klassen *android.test.MoreAsserts* und *android.test.ViewAsserts* implementiert. Die Methoden der MoreAsserts-Klasse stellen eine erweiterte Liste der Assert-Methoden aus dem JUnit-Framework dar. Als Ergänzung stellt die *ViewAsserts*-Klasse Methoden bereit, die speziell für Benutzeroberflächen und die Interaktion mit dem Benutzer ausgelegt sind.

- Instrumentation

  Für das Testen der Interaktion mit der Android-Anwendung kann die Instrumentation API genutzt werden. Dabei handelt es sich um Klassen, die von der *TestCase*- Klasse aus dem JUnit-Framework erben. Die *InstrumentationTestCase*-Klasse bietet die Möglichkeit, die Reaktion der Anwendung auf Tastatureingaben, Veränderung der Bildschirmausrichtung oder das Verhalten von Elementen der Benutzeroberfläche zu überprüfen.

- Mock-Objekte

  Mock-Objekte bieten die Möglichkeit, isolierte Objekte der zu testenden Klasse zu erzeugen. Dieses Verfahren ist notwendig, wenn Tests auf Methoden ausgeführt werden, die Abhängigkeiten zu anderen Ressourcen haben. Als Beispiel kann der Aufruf einer Test-Methode sein, die eine Datenbankveränderung hervorrufen würde. Da die Datensätze durch die Test-Methoden nicht verändert werden dürfen, wird ein Mock-Objekt erzeugt, welches isoliert von der realen Datenbank auf den Methodenaufruf reagiert. Ein Mock-Objekt stellt somit eine Art Dummy dar, der die zu testende Umgebung repräsentiert, um ohne Auswirkung auf die produktiven Ressourcen testen zu können.

- Context

  Der Context einer Android-Anwendung stellt die Ablaufumgebung mit den entsprechenden Parametern für die Funktionsfähigkeit dar. Für das Testen von Datenbank-, Datei- und Ordner-Operation werden zwei Context-Klassen angeboten:

  - Die Klasse *IsolatedContext* bietet eine isolierte Systemumgebung an, in dem die durchzuführenden Operationen keinen Einfluss auf das reale Dateisystem haben, d.h. die Daten bleiben unverändert, weil sie in einem Testbereich des Dateisystems verarbeitet werden.
  - *RenamingDelegatingContext* bietet einen eingeschränkten isolierten Kontext an, in dem die Datei- und Datenbankzugriffe durch einen IsolatedContext abgebildet werden. Alle anderen Systemaufrufe werden durch den realen Kontext verarbeitet.

Die folgenden Kapitel sollen kurz aufzeigen, welche Möglichkeiten es gibt, Android-Anwendungen zu testen.

2.3.3.2 Activity Testing

Die Klasse *Activity* stellt die Grundlage für eine Android-Anwendung mit Benutzeroberflächen dar, mit denen der Benutzer interagieren kann. Eine Anwendung kann aus mehreren Activities bestehen, die sich gegenseitig starten können, wenn die entsprechenden Berechtigungen bestehen. Für das Testen von Activities, steht die bereits erwähnte Instrumentation API (s. *2.3.3.1 Grundlagen*), mit der Klasse *InstrumentationTestCase* zur Verfügung. Folgende Hauptfunktionen sollen durch die Klasse abgedeckt werden:

- Kontrolle des Lebenszyklus

  Android-Anwendungen durchlaufen verschiedene Lebenszyklen. Dazu zählen unteranderem das Starten der Anwendung, das Pausieren, wenn andere Applikationen aufgerufen werden, das Zurückkehren von anderen Anwendungen und das Beenden der Anwendung. Für jeden dieser Fälle stehen in einer Android besondere Methoden zur Verfügung (*onCreate()*, *onPause()*, *onDestroy()*, *onResume()*, *onStop()*), die bei der entsprechenden Veränderung des Anwendungszustandes aufgerufen werden. Die Instrumentation API bietet die Möglichkeit, die unterschiedlichen Zustände

der Anwendung zu erzwingen und zu überprüfen, ob das gewünschte Verhalten auf den eingetretenen Anwendungszustand eingetreten ist.

- Einbinden von Abhängigkeiten

  Die Instrumentation API bietet Funktionalitäten an, um Abhängigkeiten einer Anwendung zu anderen Ressourcen abzubilden. So besteht die Möglichkeit, kritische Ressourcen durch Mock-Objekte zu ersetzen oder die Anwendungen in einem isolierten Kontext zu testen.

- Benutzereingaben

  Als dritte Funktion bietet die Klasse *InstrumentationTestCase* die Möglichkeit, Benutzereingaben (Tastendruck, Berührung des Touchscreens usw.) zu simulieren und die Reaktion der Anwendung auf die Eingaben zu überprüfen.

Als Oberklassen für die eigenen Test-Klassen stehen drei Klassen zur Verfügung, die die Klasse *InstrumentationTestCase* implementieren. Diese Klassen unterscheiden sich vor allem in der Umgebung, in der die Tests der Anwendung ablaufen.

- *ActivityInstrumentationTestCase2*

  Diese Klasse stellt Funktionalitäten bereit, um mehrere Activities in einer Anwendung zu testen. Für die durchzuführenden Tests wird eine Instanz, der zu testenden Anwendung, in der normalen System-Umgebung generiert. Für den Aufruf anderer Activities nutzbare Mock-Intents, können im Gegensatz zu den Mock-Objekten für den System-Kontext, eingesetzt werden.

- *ActivityUnitTestCase*

Im Gegensatz zu der Klasse *ActivityInstrumentationTestCase2* wird bei der Verwendung der Oberklasse *ActivityUnitTestCase* ausschließlich eine Activity in Isolation getestet. Aus diesem Grund ist auch die Einbindung von Mock Objekten für den System-Kontext möglich. Die Nutzung von Mock-Objekten anderer Activities ist nicht möglich.

- *SingleLaunchActivityTestCase*

  Die Klasse *SingleLaunchActivityTestCase* wird für Testumgebungen genutzt, die sich während des Tests nicht verändern. Die zugehörigen *setUp()* und *tearDown()* Methoden werden nur einmal aufgerufen. Somit bleibt die Testumgebung für die durchzuführenden Tests gleich und es kann gezielt nachvollzogen werden, was bei einem mehrfachen Aufruf der gleichen Activity passiert. In dieser Testumgebung sind keine Mock-Objekt zulässig.

2.3.3.3 Service Testing

Services sind Komponenten von Android-Anwendungen, die eine länger andauernde Aufgabe erfüllen und keine Benutzeroberfläche bzw. keine Interaktion mit dem Benutzer erfordern. Services laufen im gleichen Prozess ab, wie die Anwendung durch die sie gestartet wurden. Somit dürfen die Services einer Android-Anwendung nicht mit Threads und Systemprozessen verwechselt werden. Ähnlich wie die Activities durchlaufen auch Services einen bestimmten Lebenszyklus, für dessen Steuerung diverse Methoden zur Verfügung stehen (*onCreate()*, *onDestroy()*, *onStartCommand()*). Folgende Klasse stellt die Grundlage für das Testen von Services bereit.

- *ServiceTestCase*

  Die Klasse *ServiceTestCase* ist eine Unterklasse der Klasse *AndroidTestCase* aus dem JUnit-Framework. Die Klasse stellt Methoden zur Verfügung, um die Testumgebung zu initialisieren. Weiterhin können Mock-Objekte von Anwendungen (*setApplication()*)und Kontexten (*setContext()*) eingebunden werden, die die Testumgebung von dem realen zu testenden System isolieren. Die Initialisierung der Testumgebung wird solange herausgezögert, bis die Methode *ServiceTestCase.startService()* oder *ServiceTestCase.bindService()* aufgerufen wird.

2.3.3.4 Content Provider Testing

Content Provider sind für die tabellarische Speicherung lokaler Daten einer Anwendung und zur Bereitstellung dieser Daten für andere Anwendungen zuständig. Neben den systemeigenen Content Providern, wie den Kontakt-Listen, können auch eigene Provider implementiert werden. Für das Testen dieser Provider steht eine Basisklasse zur Verfügung.

- *ProviderTestCase2*

  Die Klasse *ProviderTestCase2* stellt eine Unterklasse von *AndroidTestCase* dar. Die Initialisierung der Testumgebung spielt bei den Content Providern eine wichtige Rolle, weil die durch den Provider verwalteten Daten durch die Tests nicht beeinflusst bzw. verändert werden dürfen. Aus diesem Grund wird durch den Konstruktor ein isolierter *IsolatedContext* generiert, der Datei- und Datenbank-Operation erlaubt, aber andere Interaktionen stellvertretend für das reale Android-System abwickelt. Weiterhin wird durch den Konstruktor ein Mock-Objekt des *ContentResolvers* erzeugt, der die bereitgestellten Daten entgegen nimmt. Abschließend wird ein Objekt der Klasse *ContentProvider* erzeugt, dass durch die vorherige Initialisierung in einer isolierten Testumgebung abläuft.

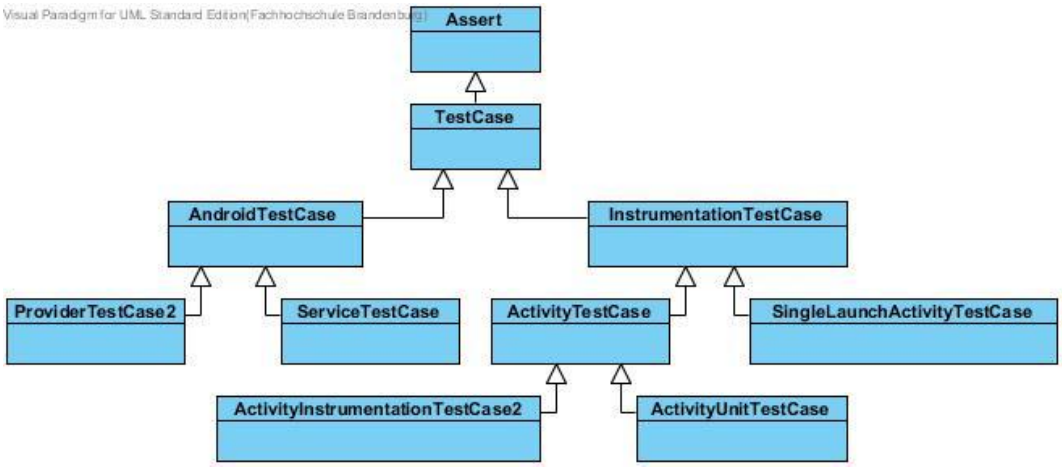

Abbildung 2: Übersicht der Test-Klassen

Abbildung 2 fasst die Ausführungen zu den Test-Klassen nochmals grafisch zusammen.

3 Analyse „speedikon® DAMS“

Im folgenden Teilabschnitt wird die zugrundeliegende Anwendung „speedikon® DAMS“ näher beschrieben, wobei kurz auf die Funktionalitäten der Web-Anwendung und die derzeitige Verwendung im Verlag eingegangen wird. Neben der Erarbeitung von Anwendungsfällen für eine portable Version, soll das Hauptaugenmerk auf die Datenbankstruktur der Anwendung gelegt werden.

3.1 Funktionen der Web-Anwendung

Die Web-Anwendung „Datacenter Center Asset Management Solutions“ (DAMS) der Firma „speedikon®“ bietet diverse Möglichkeiten zur Verwaltung und Dokumentation von Bestandsdaten in Rechenzentren an, die nachfolgend kurz erläutert werden sollen. (vgl. [Dam11])

- Verwaltung der Bestandsdaten

  Die Hauptaufgabe, die durch DAMS abgebildet wird, ist die Verwaltung von Bestandsdaten. Zu diesen Bestandsdaten können unteranderem Angaben zu Servern, zu Netzwerk- und Peripherie-Geräten sowie ähnlichen Komponenten gehören. Für jedes einzupflegende Gerät müssen diverse Informationen vorgehalten werden. Neben technischen Informationen, wie die Stromaufnahme und Wärmeabgabe eines Gerätes, können auch Netzwerk-Angaben, wie IP- und MAC-Adresse, registriert werden. Weiterhin besteht die Möglichkeit, kaufmännische Angaben, wie Inventar- und Seriennummern oder ähnliche Informationen, zu speichern. Durch die große Vielfalt solcher gerätespezifischen Daten, kann DAMS als zentrale Informationsquelle für die Bestandsdaten genutzt werden.

- Dokumentation und Visualisierung

  Der zweite wichtige Aspekt, der mit DAMS abgebildet werden kann, ist die Dokumentation und Visualisierung der Bestandsdaten. Die Dokumentation hilft dabei, die Standorte an denen die Geräte verbaut sind, genauer zu beschreiben und festzuhalten. Einerseits besteht so die Möglichkeit, die verbauten Geräte zeitnah mit Hilfe einer Suchfunktion wiederzufinden. Andererseits kann durch die lückenlose Dokumentation die Planung und der Einsatz neuer Geräte unterstützt werden, um freie Standorte für künftig zu verbauende Komponenten ausfindig zu machen. Ergänzend zur eigentlichen Dokumentation kann eine 2D- und 3D-Visualisierung der Re-

chenzentren erfolgen. Das sorgt für einen hohen Wiedererkennungswert der Rechenzentren und unterstützt zusätzlich die Suche von Komponenten.

- Kabeldokumentation

  DAMS stellt Funktionen bereit, um die gesamte Netzwerk-Verkabelung der verbauten Geräte zu dokumentieren, so dass komplette Verschaltungspläne der Komponenten vorliegen und die internen Abhängigkeiten der einzelnen Geräte aufgezeigt werden können. Neben den Netzwerkkabeln können auch die Kabel der Stromversorgung in ähnlicher Form dokumentiert werden.

- Reports

  Mit Hilfe von DAMS ist es möglich, diverse Auswertungen in Form von Berichten zu generieren. Für die langfristige Planung der Rechenzentren kann diese besondere Form der Auswertungen insbesondere dazu genutzt werden, um einen gezielten Überblick sowohl über freie Ressourcen, als auch die aktuelle Auslastung der Rechenzentren zu erhalten.

3.2 Verwendung in der Projektumgebung

Derzeit wird DAMS im Verlag genutzt, um die Dokumentation der Bestandsdaten zu unterstützen. Dabei werden die in Abschnitt *3.1 Funktionen der Web-Anwendung* beschriebenen Gerätedaten und Kabelwege aufgenommen. Zur Unterstützung der Kabeldokumentation sind an sämtlichen Netzwerk-Kabeln eindeutige Inventarnummern mit entsprechendem Strichcode angebracht worden. Für die Stromkabel werden ähnliche Nummerierungen verwendet. Die Dokumentation der Geräte-Standorte und der dazugehörigen Kabelwege ist in den Rechenzentren weitgehend abgeschlossen. In einem weiteren Schritt wird die Vervollständigung der gerätespezifischen Bestandsdaten erfolgen. Die Dokumentation der Stromkabel wird aufgrund der geringen Priorität zu einem späteren Zeitpunkt durchgeführt.

In *Abbildung 3* ist eine schematische Darstellung gegeben, anhand derer die Einbettung der Web-Anwendung „speedikon® DAMS“ in die technische Projektumgebung näher erläutert werden soll.

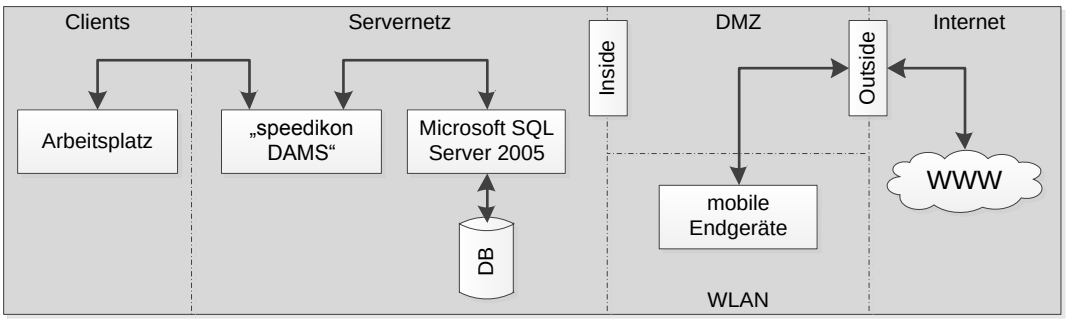

Abbildung 3: schematische Darstellung der Projektumgebung

Die Web-Anwendung ist auf einem separaten Server installiert. Die Datenbank für die Speicherung der Daten wird durch einen abgesetzten Server gehalten, der den Zugriff auf die Datenbank realisiert. Diese Struktur ermöglicht den Zugriff auf die Daten durch andere Anwendungen, ohne die Verwendung von „speedikon® DAMS“. Weiterhin wird deutlich, dass der derzeitigen WLAN-Architektur, aufgrund unternehmensinterner Sicherheitsrichtlinien, kein Zugriff auf das interne Servernetz ermöglicht wird. Die Funktionalitäten des DAMS-Systems werden den Clients über ein Web-Frontend zur Verfügung gestellt.

3.3 Analyse der Datenbank

In diesem Teilabschnitt soll kurz der Aufbau der zugrundeliegenden Datenbank näher betrachtet werden, in der die Datensätze der Web-Anwendung vorgehalten werden. Grundlage der Datenbank ist ein Microsoft® SQL Server® 2005. Die Analyse des Datenbankschemas wird durch fehlende Schlüsselbeziehungen, die im DBMS nicht erkenntlich sind, deutlich erschwert. Aus diesem Grund ist nachfolgend ein Entity-Relationship-Diagramm skizziert in dem die wichtigsten Tabellen, einschließlich ihrer Beziehungen, zu sehen sind.

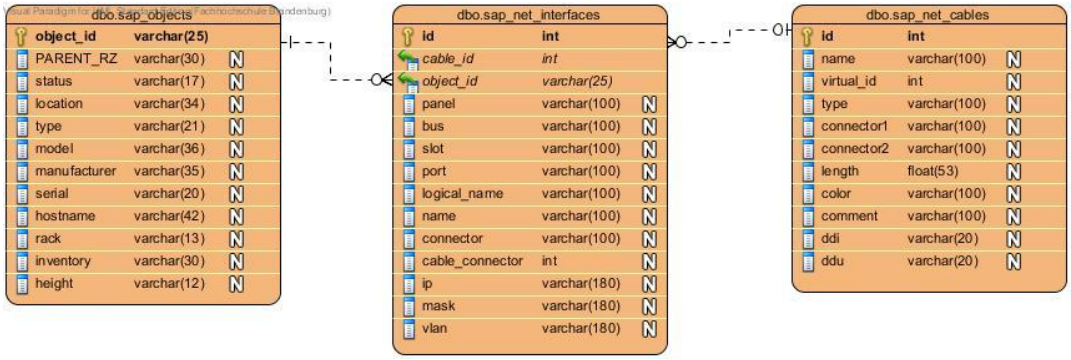

Abbildung 4: Ausschnitt der Datenbank

Die Analyse der Tabelle *dbo.sap_objects* ist nicht trivial, da die Spalten als Bezeichnung ausschließlich eine laufende Nummerierung besitzen (A00-A792). Aus diesem Grund ist in der *Abbildung 4* nur ein Ausschnitt dieser Tabelle zu sehen, in der die entsprechenden Spalten eindeutig bezeichnet wurden. Die Spalte *PARENT_RZ* ist die einzige Spalte der Tabelle *dbo.sap_objects*, die im originalen Datenbank-Schema einen eindeutigen Namen trägt.

Die drei dargestellten Tabellen stellen die benötigten Datensätze für den Use-Case „Suche starten“ bereit. Die Tabelle *dbo.sap_objects* stellt dabei die zentrale Tabelle der Web-Anwendung „speedikon® DAMS“ dar. In dieser Tabelle werden alle Inventardaten mit ihren entsprechenden Attributen gespeichert. Die Tabelle in der Mitte der Abbildung (*dbo.sap_net_interfaces*) speichert dagegen die Daten sämtlicher Netzwerkschnittstellen, die in der Anwendung verwaltet werden. Die Spalte *object_id* referenziert die Spalte *object_id* aus der Tabelle *dbo.sap_objects*, um die Beziehung zwischen dem Gerät und der verbauten Netzwerkschnittstelle herzustellen. Weiterhin steht die Spalte *cable_id* in Beziehung zu der Spalte *id* der Tabelle *dbo.sap_net_cables*, in der die Informationen zu den verwalteten Netzwerk-Kabeln vorgehalten werden. Diese Beziehung zwischen den Tabellen ist wichtig, um die Use-Cases „Kabel suchen“ und „Verkabelungswege verfolgen“ realisieren zu können.

Neben den nicht erkennbaren Schlüsselbeziehungen im abgeleiteten Datenbankschema stellen auch die Datensätze einige Probleme dar. Für ein Gerät können, bedingt durch Umzüge in verschiedene Rechenzentren, mehrere Datensätze in der Datenbank vorhanden sein. Das führt dazu, dass eine Suche nach eindeutigen Parametern, wie z.B. Inventarnummer oder Seriennummer, zu mehreren Ergebnissen führt. Dieses Verhalten muss bei der Implementierung der Datenbank-Abfragen bzw. bei der Ausgabe in der mobilen Anwendung Berücksichtigung finden. Ein weiteres Problem stellt die Kabelverfolgung dar, weil die Beziehungen mehrerer Netzwerkkabel, die logisch zu einer Netzwerkverbindung gehören, aus der Tabelle nicht ableitbar sind.

3.4 Konzeption der Anwendungsfälle

Nachfolgend werden mit einem Use-Case-Diagramm (s. *Abbildung 5)* mögliche Anwendungsfälle für eine mobile Anwendung aufgezeigt.

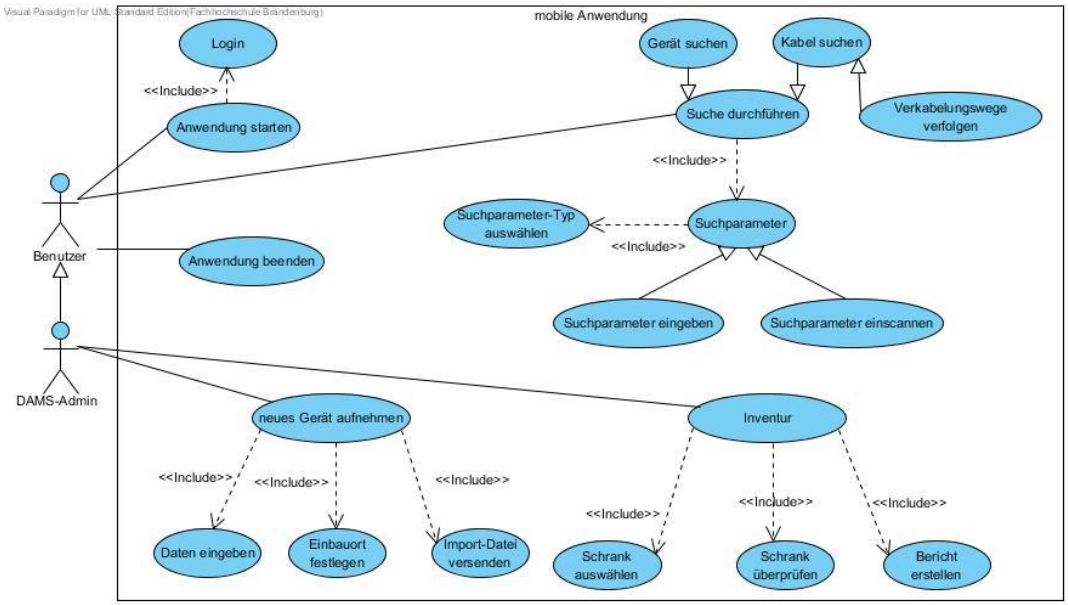

Abbildung 5: Mögliche Anwendungsfälle der mobilen Applikation

Für die mobile Anwendung wird von zwei unterschiedlichen Akteuren mit verschiedenen Aufgabengebieten ausgegangen. Der Akteur „Benutzer“ stellt einen Anwender mit eingeschränkten Benutzerrechten dar, der nur die Suchfunktionen der Anwendung nutzen soll. Diese eingeschränkte Benutzerrichtlinie ist an die derzeitige Web-Anwendung angelehnt, in der die Mitarbeiter nur die gewünschten Suchfunktionen nutzen dürfen. Diese Funktionen sollen die Geräte- und Kabel-Suche mit Hilfe unterschiedlicher Suchkriterien ermöglichen. Dabei soll diesem Anwender ermöglicht werden, den Suchparametertyp auszuwählen, den Parameter einzugeben oder einzuscannen und die Suche zu starten. Die grundlegenden Funktionen wie das Starten mit den entsprechenden Login-Daten der Web-Anwendung und das Beenden der mobilen Anwendung stellen wesentliche Bestandteile der Realisierung dar.

Die Hauptaufgabe des Akteurs „DAMS-Admin“ dagegen ist es, die Aktualität der Datensätze in der Web-Anwendung zu gewährleisten. Für diesen Aspekt sind in der mobilen Applikation weitere unterstützende Anwendungsfälle vorzusehen. Vorstellbar wären hier die in *Abbildung 5* genannten Use-Cases „neues Gerät aufnehmen“ oder „Inventur“ mit ihren Unteraufgaben, die unterstützend für die Aktualität der Daten dienen können.

Aufgrund der fehlenden Schnittstellen zu den Daten der Web-Anwendung soll für die prototypische Implementierung ausschließlich der lesende Zugriff auf die Datenbank betrachtet werden. Aus diesem Grund sollen ausschließlich die Anwendungsfälle der Gruppe „Benutzer“ in der weiteren Realisierung Beachtung finden. Nachfolgend soll kurz der Anwendungsfall „Suche starten“ in tabellarischer Form näher erläutert werden, um die Grundlage für die Implementierung der Hauptfunktion dieses Prototyps zu schaffen.

Tabelle 1: Use-Case-Beschreibung "Suche starten“

Name	Suche starten
Kurzbeschreibung	Ermöglicht den Abruf von Bestandsdaten mit Hilfe verschiedener Such-Parameter
Akteure	Benutzer, DAMS-Admin
Auslöser	Aufruf der Suche durch einen Akteur
Vorbedingung	Die Netzwerk-Verbindung zur Datenbank ist vorhanden.
Ergebnis	Anzeige der gewünschten Inventardaten
Nachbedingung	Es kann eine erneute Suche durchgeführt werden.
Essenzieller Ablauf	1. Aufruf der Suchfunktion 2. Auswahl der gewünschten Suche (Gerätesuche oder Kabelverfolgung) 3. Auswahl des Such-Parameter-Typs 4. Eingabe des Suchbegriffs (manuell oder einscannen) 5. Ausgabe der Bestandsdaten

Eine nähere Betrachtung der Anwendungsfälle „Anwendung starten“ und „Anwendung beenden“ erfolgt aufgrund des geringen Funktionsumfangs nicht.

4 Umsetzung

4.1 Realisierung des Web-Service

4.1.1 Zieldefinition

Die vorhandene Web-Anwendung „speedikon® DAMS“ stellt keine definierten Schnittstellen für den Austausch der vorhandenen Daten mit anderen Anwendungen zur Verfügung. Aus diesem Grund soll für die prototypische Implementierung der mobilen Anwendung eine Schnittstelle auf Basis eines Web-Service geschaffen werden. Ziel soll es sein, die Login- und Suchanfragen der Benutzer zu verarbeiten und die Ergebnisse in Form von JSON-Objekten zurückzuliefern.

4.1.2 Planung

In *Abbildung 6* wird die geplante Funktionsweise des Web-Service schematisch beschrieben. Dieses Diagramm bildet die Grundlage, um für die Implementierung benötigte Komponenten ableiten zu können. Als Einstiegspunkt für den Benutzer wird der Anmeldeversuch an der mobilen Applikation angenommen. Die eingegebenen Anmeldedaten werden an den Web-Service übermittelt. Dieser muss Funktionalitäten bereitstellen, um die, durch die Web-Anwendung verwaltete, Benutzerliste einzulesen und die übergebenen Benutzerdaten zu vergleichen. Das Ergebnis des Anmeldeversuchs ist in Form eines JSON-Objektes an die Applikation zu übertragen. Eine positiv verlaufene Anmeldung ermöglicht dem Benutzer in der mobilen Anwendung die Suchfunktionen zu nutzen. Aus diesem Verhalten ergeben sich weitere, durch den Web-Service abzudeckende Ereignisse. Anhand eines Suchkriteriums, fordert der Benutzer den entsprechenden Datensatz an. Der Web-Service nimmt diese Suchanfrage entgegen. Bei der Verarbeitung muss unterschieden werden, ob Gerätedaten bereitgestellt oder eine Kabelverfolgung durchgeführt werden soll. Für beide Funktionen sind entsprechende Methoden zu implementieren. Zudem sind daneben noch Methoden für den Zugriff auf die Datenbank zu realisieren.

Wie in der Betrachtung der Datenbank beschrieben (s. *3.3 Analyse der Datenbank*), kann eine Abfrage eine Liste von gefunden Datensätzen generieren. In diesem Fall muss die Anwendung Funktionen bereitstellen, die die Verarbeitung mehrerer Ergebnisse erlaubt und diese dann an die mobile Anwendung übergeben kann. Zusätzlich müssen Funktionen implementiert werden, die eine Abfrage von Datensätzen anhand eindeutiger Merkmale (z.B. Primärschlüssel) ermöglicht.

Für eine Such-Anfrage ohne Ergebnis ist eine entsprechende Fehlermeldung zu generieren.

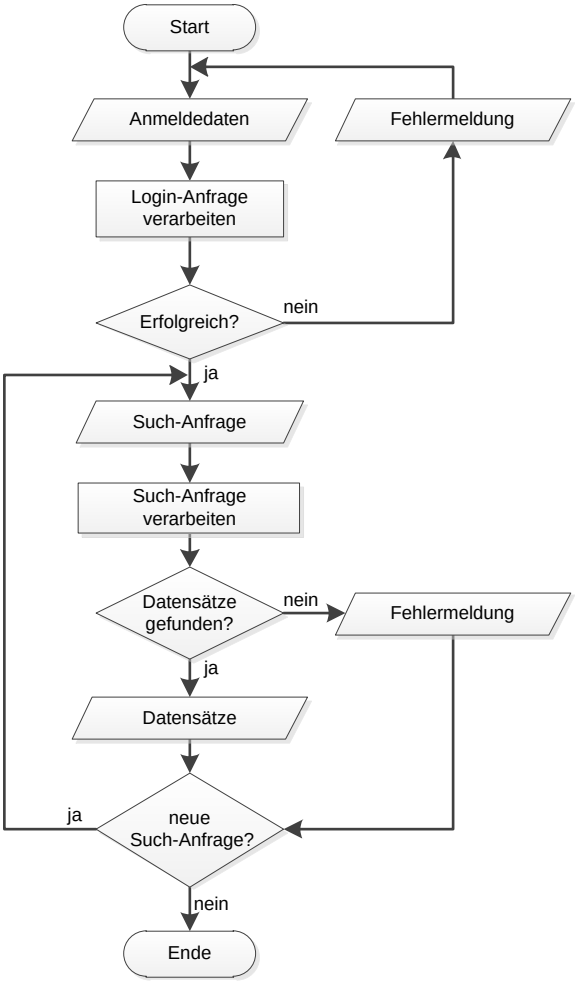

Abbildung 6: Ablaufdiagramm des Web-Service

Als Programmiersprache für den Web-Service soll Java™ unter Verwendung der Entwicklungsumgebung „Eclipse“ genutzt werden. Für die Umsetzung soll weiterhin das Framework „Jersey“ und die „Java™ Persistence API“ (JPA), für den Zugriff auf die Datenbank, Verwendung finden. Als Applikations-Server wird ein Apache Tomcat in der Version 7 zum Einsatz kommen.

4.1.3 Umsetzung

Nachfolgend wird mit Hilfe eines Klassendiagramms die Struktur und Umsetzung des Web-Service näher beschrieben.

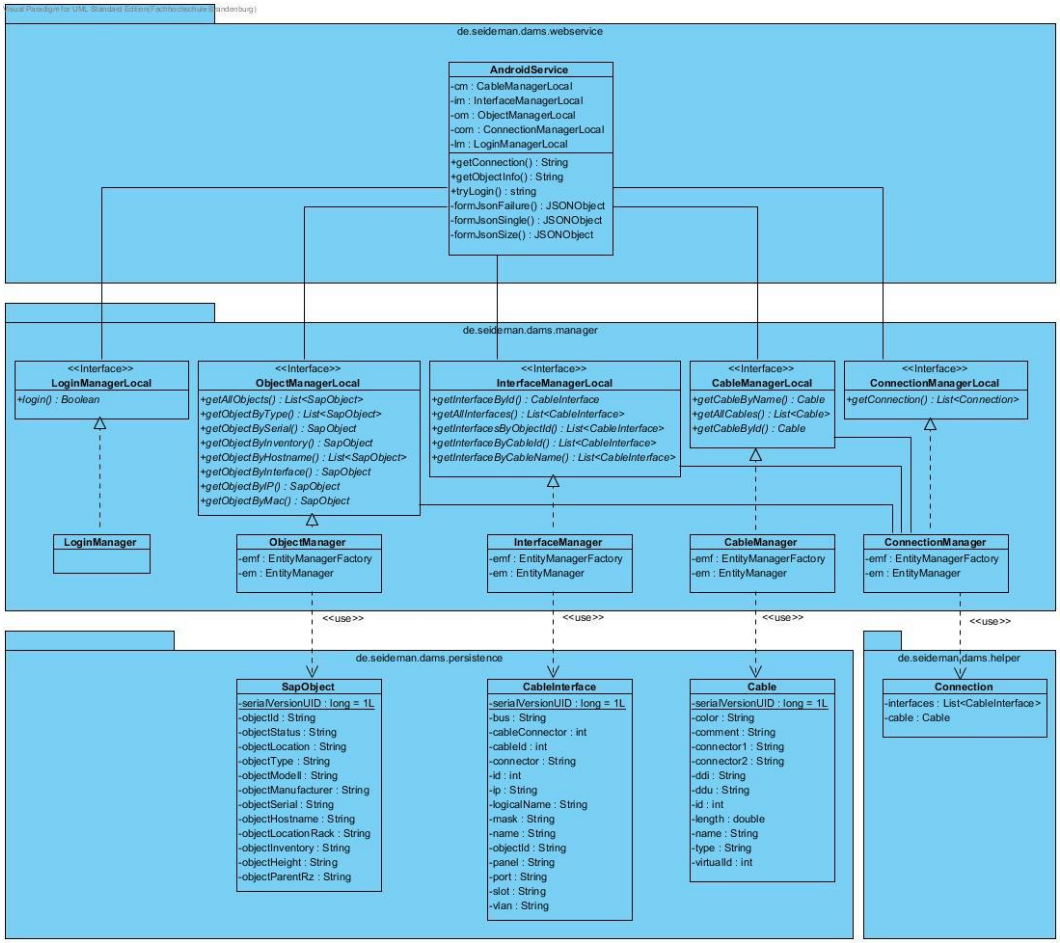

Abbildung 7: Klassendiagramm des Web-Service

Die Struktur des Web-Service ist in die vier Pakete *webservice*, *manager*, *persistence* und *helper* unterteilt.

- *webservice*

  Die Schnittstelle zwischen der mobilen Applikation und des Web-Service stellt die Klasse *AndroidService* im Paket *de.seideman.dams.webservice* dar. Mit Hilfe dieser Klasse werden der mobilen Anwendung die Funktionalitäten des Web-Services zur Verfügung gestellt. Die Methode *getObjectInfo()* implementiert den Zugriff auf die gewünschten Inventardaten. In ähnlicher Form ermöglicht *getConnection() die* Abfrage zur Verfolgung der Netzwerk-Verkabelung an. Die Methode *tryLogin()* bietet der mobilen Anwendung die Möglichkeit, Anmelde-Versuche durch den Benutzer zu verarbeiten.

Neben den Methoden zur Bereitstellung der Funktionalitäten sind in der Klasse *AndroidService* die Methoden *formJsonSize(), formJsonSingle()* und *formJsonFailure()* für die Umformung der entsprechenden Datensätze in das Dateiaustauschformat JSON implementiert. Die Methode *formJsonSize()* erzeugt ein JSON Objekt, in dem die Anzahl der gefunden Datensätze und einige Werte jedes Datensatzes übergeben werden. Diese Implementierung bietet dem Benutzer in der mobilen Anwendung die Möglichkeit eine Auswahl zu treffen, wenn seine Suche mehrere Ergebnisse geliefert hat. Durch die Auswahl des Benutzers in der Applikation wird das eigentliche Objekt mit Hilfe der Methode *formJsonSingle()* erzeugt. Wenn bei der Such-Anfrage kein passender Datensatz gefunden wird, erzeugt die Methode *formJsonFailure()* eine JSON-Objekt, dass die entsprechende Fehlermeldung beinhaltet.

- *manager*

  Die Programmlogik ist im Paket *de.seideman.dams.manager* zusammengefasst. Auf die Methoden, die durch die Klasse *AndroidService* benötigt werden, kann mit Hilfe der entsprechenden Interfaces zugegriffen werden. Die Nutzung von Interfaces ermöglicht somit die Trennung der Methoden-Implementierung von der des Methoden-Aufrufs. Die in den Interfaces bereitgestellten Methoden, werden durch die gleichnamigen Klassen (ohne Namenserweiterung „Local“) implementiert. Die Klasse *LoginManager*, liest die originale Datei der Web-Anwendung ein, in der die Anmelde-Namen und die Passwörter (als MD5-Hash) der zugelassen Benutzer gespeichert sind. Die Login-Anfragen werden durch die Klasse *AndroidService* an die Klasse *LoginManager* über das Interface *LoginManagerLocal* weitergegeben. Die Klasse wertet die zuvor gefilterten Benutzer-Einträge aus und gibt den entsprechenden Wahrheitswert eines positiven oder negativen Anmelde-Versuchs zurück.

  Die Klassen *ObjectManager*, *InterfaceManager* und *CableManager* repräsentieren die Schnittstellen für den Zugriff auf die Datenbank und stellen Methoden bereit, um Objekte mit Hilfe bestimmter Suchkriterien aus der Datenbank abzufragen. Als Ergebnis dieser Abfragen werden Listen zurückgeliefert, die die gesuchten Inventardaten beinhalten. Die Klasse *ConnectionManager* implementiert die Methoden für den Anwendungsfall „Verkabelungswege verfolgen“. Aufgrund der nicht sofort ersichtlichen

Beziehungen von mehreren Netzwerkkabeln untereinander, wurde hier ein eigener Algorithmus entworfen. Ergebnis dieser Suche ist eine Liste mit Objekten der Klasse *Connection* (s. Paket *helper*).

- *persistence*

  Die Inventardaten werden in Persistenz-Objekten gespeichert. Diese Objekte stellen eine objektorientierte Repräsentation der relationalen Datenbank-Tabelle dar, d.h. jedes erzeugte Persistenz-Objekt entspricht einem Datensatz (eine Zeile) in der dazugehörigen Datenbank-Tabelle. Die benötigten Persistenz-Klassen *SapObject*, *CableInterface* und *Cable* wurden durch die Entwicklungsumgebung generiert und werden im Paket *de.seideman.dams.persistence* vorgehalten.

- *helper*

  Das Paket *de.seideman.dams.helper* beinhaltet Klasse *Connection*. Diese Klasse wird unterstützend für die Implementierung der Kabelverfolgung benötigt. Dabei wird der Ansatz verfolgt, dass jede Verbindung aus einem Kabel besteht, das mit höchstens zwei Netzwerk-Schnittstellen verbunden sein kann. Als Ergebnis der Methode *getConnection()* in der Klasse *ConnectionManager* entsteht somit eine Liste mit Objekten der Klasse *Connection,* die die Verbindung zwischen zwei Geräten repräsentiert.

4.2 Realisierung der Android-Anwendung

4.2.1 Zieldefinition

Das Ziel der exemplarisch zu entwickelnden Android-Anwendung ist es, die vorhandenen Daten der Web-Anwendung „speedikon® DAMS“ aus der abgesetzten Datenbank abzufragen und dem Benutzer in geeigneter Weise bereitzustellen. Die Anwendung soll die Anwendungsfälle des Akteurs „Benutzer“ abbilden. Als Ergänzung zu der manuellen Eingabe von Suchparametern soll die Einbindung der Barcode-Anwendung „Zxing“ realisiert werden.

4.2.2 Planung

In diesem Abschnitt wird die Planung der mobilen Anwendung aufgezeigt. Im ersten Schritt werden die benötigten Funktionsweisen der Applikation näher beschrieben.

Als Einstiegspunkt wird der Start der Anwendung angenommen. Die Applikation muss, nach vorheriger Eingabe der Benutzerdaten durch den Benutzer, den Anmeldeversuch verarbeiten. Zuvor soll eine Überprüfung der Erreichbarkeit des Web-Service erfolgen, wobei die fehlende Verbindung zum Web-Service, sowie leere Eingabe-Felder und ein negativer Anmeldeversuch dem Benutzer anzuzeigen sind. Eine erfolgreiche Anmeldung soll den Benutzer in das Hauptmenü der Anwendung führen. Entsprechend der erarbeiteten Anwendungsfälle (s. *3.4 Konzeption der Anwendungsfälle*) soll dem Benutzer ermöglicht werden, Suchkriterien auszuwählen und den Suchparameter einzugeben.

Die Parameter sind dem Web-Service zu übergeben, wobei zuvor eine Unterscheidung des Suchkriteriums erfolgen (Objektdaten oder Kabelverfolgung) muss. Gefundene Datensätze sind dem Benutzer anzuzeigen. Dabei muss beachtet werden, dass aufgrund der Datenbankstruktur mehrere Datensätze als Ergebnis möglich sind. Aus diesem Grund soll dem Benutzer eine Vorauswahl angezeigt werden, aus der er das gewünschte Objekt auswählen und anfordern kann. Die Ausgabe der Inventardaten für den Benutzer ist auf geeignete Weise durchzuführen. Sollte kein passendes Ergebnis zurückgeliefert werden, muss eine Fehlermeldung für den Benutzer erstellt werden. Suchanfragen können beliebig oft wiederholt werden.

In *Abbildung 8* wird die Planung der mobilen Applikation anhand eines Ablaufdiagramms grafisch dargestellt.

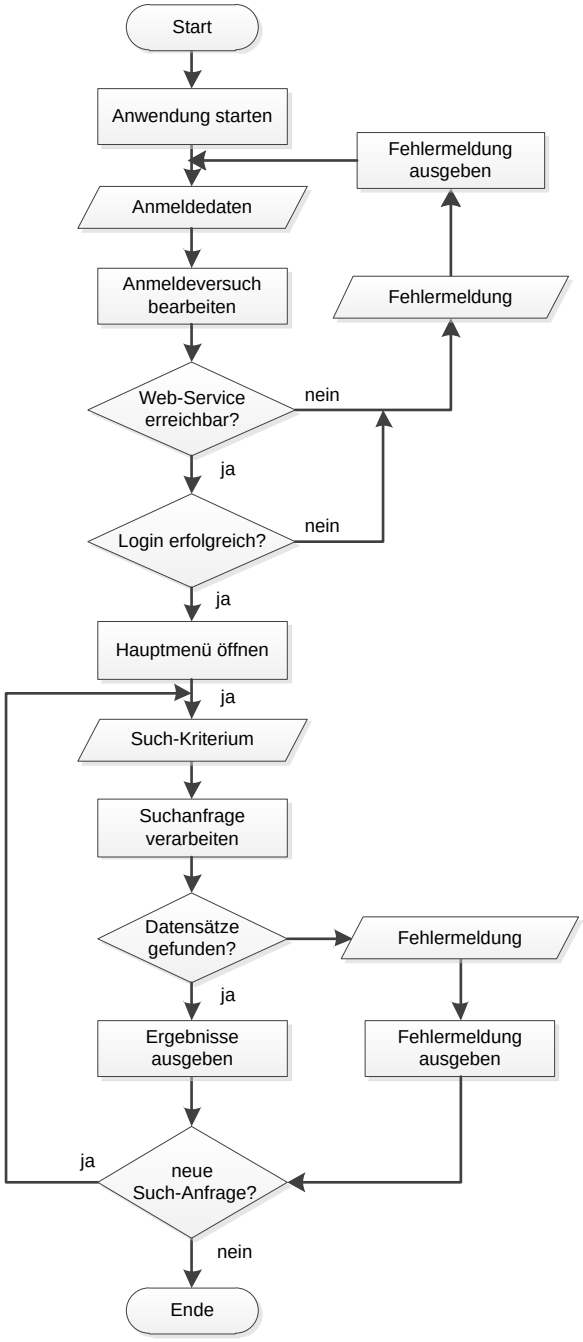

Abbildung 8: Ablaufdiagramm der mobilen Anwendung

Für die Umsetzung wurde mit Hilfe der Entwicklungsumgebung ein mögliches Layout der Anwendung erstellt (s. *Abbildung 9: Layout-Planung*). Um die grafischen Elemente zu referenzieren, wurden die Bezeichnung *#A01* (Anmeldung) und *#A02* (Hauptmenü) eingeführt. Nachfolgend erfolgt die Funktionsbeschreibung der Teilkomponenten.

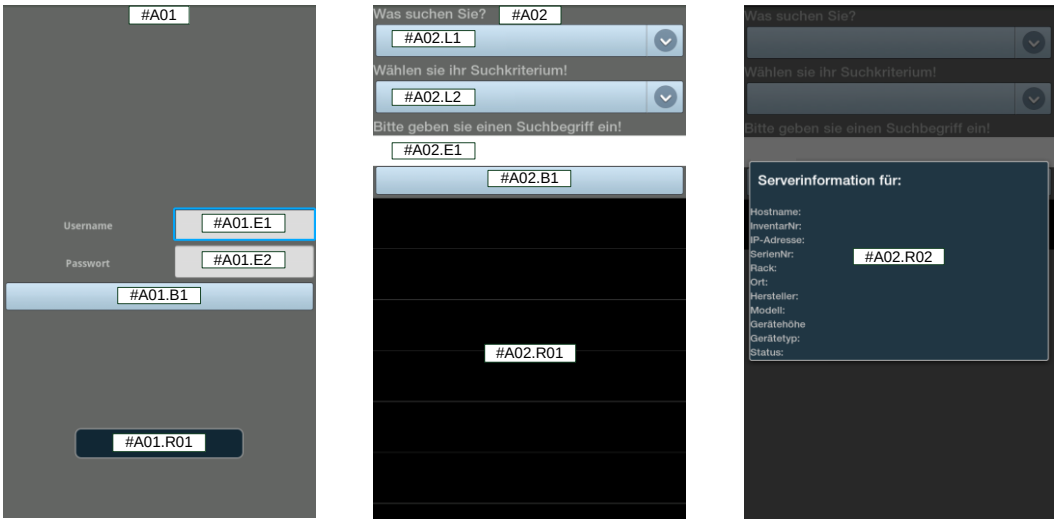

Abbildung 9: Layout-Planung

- Anmeldung *#A01*

  Die Komponenten *#A01.E1* und *#A02.E2* stellen die Eingabefelder für Benutzername und Passwort dar. Der Button *#A01.B1* soll nach Betätigung die Verarbeitung des Anmeldeversuchs starten, wobei eine Überprüfung erfolgen soll, ob die Felder #A01.E1 und *#A02.E2* leer sind. Fehlermeldungen, wie leere Eingabefelder oder ein negativer Anmeldeversuch sollen im Bereich #A01.R1 signalisiert werden.

- Hauptmenü *#A02*

  Die Teilkomponente *#A02.L1* stellt eine Auswahlliste dar. Der Benutzer kann hier entscheiden, ob Inventardaten abgefragt werden oder ob eine Kabelverfolgung durchgeführt wird. Erfolgt in *#A02.L2* die Auswahl zur Kabelverfolgung wird die Liste #A02.L2 deaktiviert, da keine weitere Einschränkung des Suchkriteriums sinnvoll ist. Wählt der Benutzer die Abfrage von Inventardaten, wird *#A02.L2* aktiviert. In diesem Fall sind in *#A02.L2* Suchkriterien anzuzeigen (z.B. Inventar-Nummer, Serien-Nummer usw.). Das Eingabefeld *#A02.E1* ist für die Eingabe des Suchbegriffs durch den Benutzer vorgesehen. Der Button *#A02.B1* startet die Verarbeitung der Such-Anfrage. Dabei erfolgt eine Überprüfung, ob das Feld *#A02.E1* leer ist. Fehlermeldungen sind in ähnlicher Form, wie bei *#A01.R1,* anzuzeigen. Die Vorauswahl der Suchergebnisse soll unter dem Button *#A02.B1* im Bereich *#A02.R1* als Liste ausgegeben werden. Durch Auswahl eines Elementes in *#A02.R1* werden die Inventardaten des entsprechenden Gerätes abgefragt. Die Ausgabe erfolgt wie in *#A02.R2*. Die

geplante Benutzeroberfläche wird nur für das Hochformat („Portrait“) erstellt.

4.2.3 Umsetzung

Nachfolgend soll mit Hilfe eines Klassendiagramms die Umsetzung und Struktur der mobilen Anwendung genauer beschrieben werden. Die Anwendung ist logisch in verschiedene Pakete aufgeteilt. Im Paket *de.seideman.dams.activities* sind die Activities (Teilanwendungen, aus denen sich die Gesamtanwendung ergibt) *Dams* und *Login* untergebracht. Das Paket *de.seideman.dams.exceptions* enthält alle die für das Abfangen von Ausnahmesituation implementierten Exception-Klassen. In diesem Fall ist es die Klasse *EmptyInputException* die generiert wird, wenn benötigte Eingabefelder leer sind.

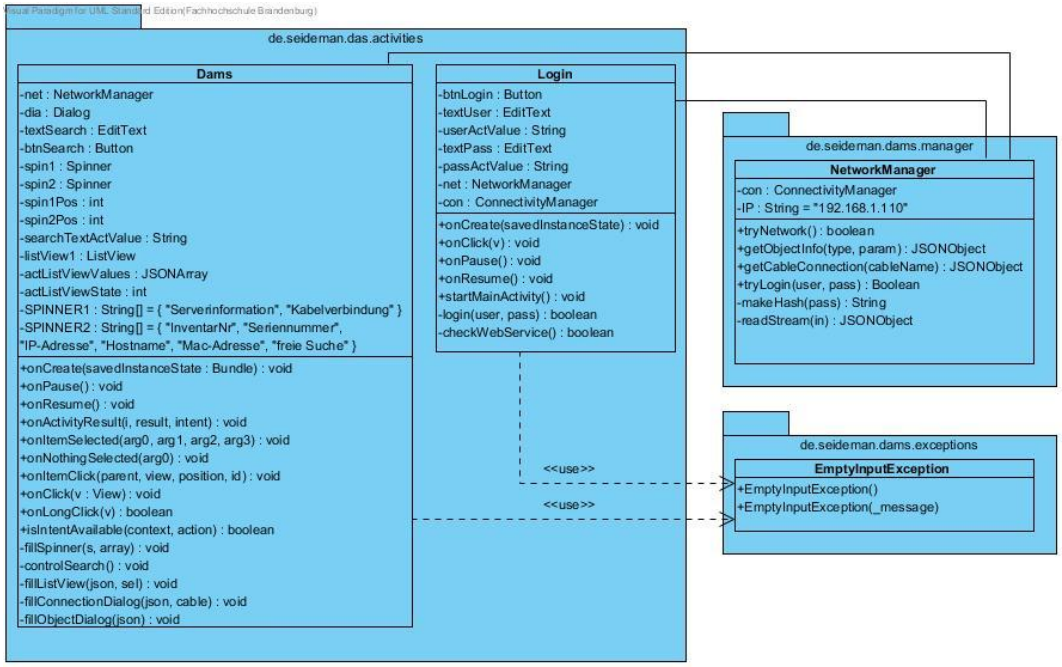

Abbildung 10: Klassendiagramm der mobilen Anwendung

Die Klasse *Login* ist für die Abwicklung von Anmelde-Versuchen durch die Benutzer zuständig und startet bei erfolgreichem Login mit Hilfe der Methode *startMainActivity()* die Hautanwendung *Dams.* Bevor der Login erfolgen kann wird die Verbindung zum Web-Service (*checkWebService()*) getestet und eine Überprüfung durchgeführt, ob die Eingabefelder leer sind.

Die Hauptfunktionalitäten der mobilen Anwendung werden durch die Klasse *Dams* unter Verwendung der Klasse *NetworkManager* realisiert. Für die Realisierung der grafischen Oberfläche sind einige Elemente notwendig, die beim Start der Anwendung initialisiert werden. Diese grafischen Elemente werden in einer Konfigurationsdatei mit den entsprechenden Parametern angelegt. Jedem dieser

Elemente muss dabei eine eindeutige Bezeichnung in der entsprechenden Konfigurationsdatei zugeordnet werden. Durch Initialisierung entsprechender Objekte in der Methode *onCreate()* ist ein Zugriff auf die grafischen Elemente zur Laufzeit möglich. Unteranderem können so Eingabewerte ausgelesen und verändert werden.

Die Verwendung von Objekten, wie Buttons, erfordert die Überwachung manueller Benutzereingaben, d.h. es werden mit Hilfe der Methoden *onClick(), onLongClick(), onItemSelected() oder onItemClick()* entsprechende Funktionalitäten implementiert. Diese Methoden werden durch Listener-Interfaces, wie dem *OnClickListener*, bereitgestellt.

Die Oberklasse *Activity* stellt Methoden für die Verwaltung des Lebenszyklus einer Android-Anwendung zur Verfügung. Das Verhalten der Anwendung ist in den Methoden *onPause()*, *onResume()* und *onActivityResult()* implementiert. In der mobilen Anwendung wird die *onActivityResult()*-Methode aufgerufen um Daten (eingescannter Strichcode) zwischen der Anwendung „Zxing[5]“ und der mobilen Applikation auszutauschen. Vor der Nutzung von „Zxing“ wird mit *isIntentAvailable()* überprüft, ob die Anwendung installiert ist. Die Methoden *onPause()* und *onResume()* speichern den Zustand der Anwendung bei kurzzeitigem Verlassen und sichern diesen bei einem Neustart der Anwendung zurück.

Neben den, durch die Oberklasse *Activity,* bereitgestellten Methoden sind weitere eigene Funktionalitäten zu implementieren. Dazu zählen in diesem Fall die Methoden *fillSpinner()*, *fillListView()*, *fillObjectDialog(), fillConnectionDialog()* und *controlSearch().* Für die Initialisierung der Auswahlfelder („Spinner“) beim Start der Hauptanwendung nach erfolgreichem Login ist die Methode *fillSpinner()* zuständig. Die Methode *controlSearch()* wertet die Benutzer-Eingaben nach Betätigen des „Suchen“-Buttons aus und stößt die entsprechenden Methoden der Klasse *NetworkManager* an. Die Ergebnisliste der Suchanfrage wird an die Methode *fillListView()* übergeben, um die Liste mit der Vorauswahl der Objekte zu generieren. Die JSON-Objekte mit den Suchergebnissen werden an die Methoden *fillObjectDialog()* oder *fillConnectionDialog()* übergeben, um die entsprechenden Ausgabe-Fenster zu erzeugen.

Die Methoden für die Kommunikation mit dem Web-Service sind in die Klasse *NetworkManager* im Paket *de.seideman.dams.manager* ausgelagert. Zur Über-

[5] http://code.google.com/p/zxing/wiki/ScanningViaIntent

prüfung der Netzwerkverbindung wird die System-Klasse *ConnectivityManager* verwendet. Realisiert wird diese Überprüfung in der Methode *tryNetwork()*. Wie bereits erwähnt, erfolgt die Steuerung der Benutzer-Anfragen durch die Methode *controlSearch()* in der *Dams*-Activity, die dann die Methode *tryLogin()*, *getObjectInfo()* oder *getCableConnection()* mit den entsprechenden Such-Parametern in der Klasse *NetworkManager* aufruft. In der Klasse *NetworkManager* werden zudem weitere private Hilfsklassen implementiert. Die Methode *makeHash()* generiert aus dem Benutzer-Passwort ein MD5-Hash, so dass das Passwort nicht im Klartext über die Netzwerk-Schnittstelle gesendet wird. Weiterhin wird die Methode *readStream()* benötigt, um die Antworten des Web-Service einzulesen.

Nachfolgendend soll an einem kurzen Quellcode-Ausschnitt die Realisierung der Kommunikation mit dem Web-Service verdeutlicht werden. Als Beispiel wird in *Listing 2* die Methode *tryLogin()* verwendet. Die Methode wird durch die Dams-Activity aufgerufen, nachdem der Benutzer den Button „Einloggen“ betätigt hat. Als Parameter werden die durch den Benutzer eingegebenen Strings aus Benutzername („user“) und Passwort („pass“) übergeben. Der String „pass“ wird beim Methoden-Aufruf durch die Methode *makeHash(pass)* konvertiert. Wichtige Bestandteile stellen die Objekte der Klassen *HttpClient* und *HttpPost* dar. Das Objekt der Klasse *HttpPost* wird mit der URI

http://"+IP+":8080/DAMS02/api/android/login

zur Methode des Web-Service initialisiert. Die Parameter werden dem *HttpPost* mit der Methode *setEntity()* als „BasicNameValuePair“ übergeben. Der Post-Request wird mit Hilfe des *HttpClient* ausgeführt und erwartet einen *HttpResponse*. Der Inhalt des *HttpResponse* wird mit Hilfe der Methode *readStream()* eingelesen und der Wahrheitswert aus dem zurückgelieferten JSON-Objekt mit der Methode *getBoolean(„login“)* extrahiert und an die aufrufende Methode zurückgegeben. Nach dem *try{}*-Block sind im originalen Quelltext noch mehrere catch()-Blöcke zu finden, die auftretenden Ausnahmesituationen auffangen und entsprechende Maßnahmen einleiten sollen. Aufgrund der Irrelevanz in diesem Beispiel wurde auf die Darstellung dieser Blöcke verzichtet.

```
public Boolean tryLogin(String user, String pass) {
        String passHash = makeHash(pass);
        Boolean result = false;
        JSONObject json = null;
        HttpClient cl = new DefaultHttpClient();
        HttpPost post = new HttpPost("http://"+IP+":8080/DAMS02/api/android/login");
        try {
                ArrayList<NameValuePair> data = new ArrayList<NameValuePair>(1);
                data.add(new BasicNameValuePair("user", user));
                data.add(new BasicNameValuePair("pass", passHash));
                post.setEntity(new UrlEncodedFormEntity(data));
                HttpResponse resp = cl.execute(post);
                HttpEntity entity = resp.getEntity();
                json = readStream(resp.getEntity().getContent());
                result = json.getBoolean("login");
        } …
return result;
}
```

Listing 2: Methode *tryLogin()* aus der Klasse *NetworkManager*

Abschließend soll mit Hilfe eines Sequenzdiagramms beispielhaft der Ablauf der Anwendung und die Kommunikationswege mit dem Web-Service skizziert werden. Als Grundlage für das Diagramm dient der Anwendungsfall „Suche durchführen“ unter Verwendung einer Seriennummer. Für ein besseres Verständnis sind in den blauen Kästchen die Begriffe „App:“ und „Web:“ aufgeführt, um die mobile Anwendung („App“) und den Web-Service („Web“) sowie deren Unterklassen klar zu unterscheiden. Aufgrund der besseren Übersichtlichkeit wird bei den Methoden-Aufrufen auf die Darstellung der Eingabe-Parameter verzichtet.

Wie in den Anwendungsfällen (s. *3.4 Konzeption der Anwendungsfälle*) bereits beschrieben hat der Benutzer die Möglichkeit den gewünschten Such-Parameter-Typ auszuwählen und das Suchkriterium einzutragen. Mit dem Button „Suchen“ wird die Methode *onClick()* in der Klasse *Dams* aufgerufen, wobei eine Überprüfung der Verbindung zum Web-Service durchgeführt wird. Besteht die Verbin-

dung, liefert die Methode *tryWebService()* der Klasse *AndroidService* ein JSON-Objekt zurück.

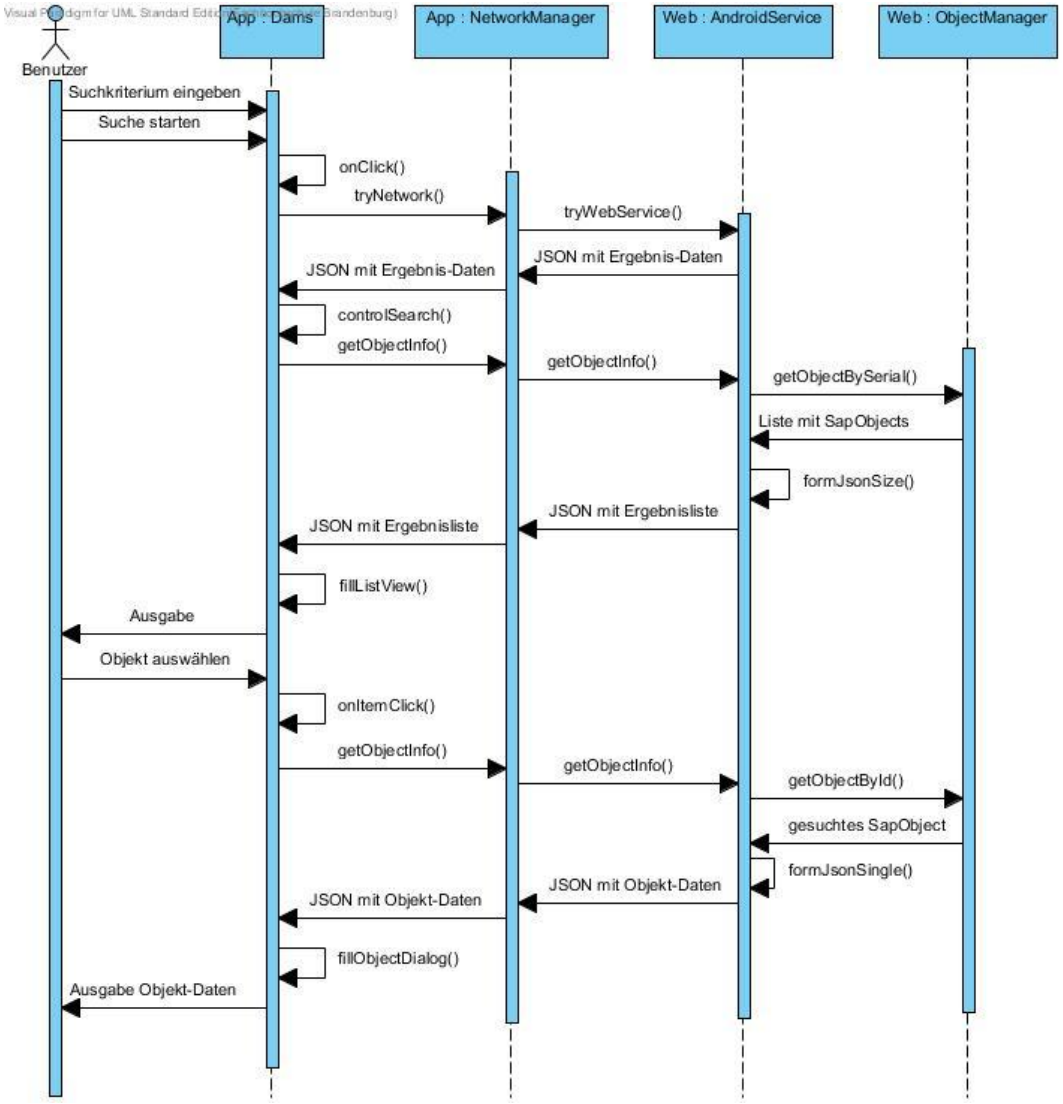

Abbildung 11: Sequenzdiagramm "Suche starten"

Das Einlesen des Such-Parameters übernimmt die Methode *controlSearch()* und ruft die Methode *getObjectInfo()* im *NetworkManager* auf. Dieser wiederum erzeugt einen *HttpPost*-Request, der die URI der entsprechenden Methode aufruft und den Such-Parameter übergibt. Die Methode *getObjectInfo()* der Klasse *AndroidService* wertet die übergebenen Parameter aus. Der vorliegende Fall zeigt eine Anfrage, in der der Benutzer eine Suche anhand der Seriennummer durchführt. Aus diesem Grund wird die Methode *getObjectBySerial()* der Klasse *ObjectManager* aufgerufen, die bei einer erfolgreichen Suche eine Liste der gefundenen Objekte erzeugt und diese über die einzelnen Instanzen an die

Dams-Activity zurückgibt. Die Ergebnisliste dient als Eingabeparameter für die Methode *fillListView(),* die eine Auswahlliste für den Benutzer der mobilen Anwendung generiert. Die Auswahl des Benutzers wird durch die Methode *onItemClick()* verarbeitet. Mit Hilfe der Methode *getObjectInfo()* der Klasse *NetworkManager* werden die Inventardaten des Objektes, durch Aufruf der Methode *getObjectById()* in der Klasse *ObjectManager,* angefordert. Die Klasse *AndroidService* formt mit Hilfe der Methode *formJsonSingle()* das JSON-Objekt und schickt es an die mobile Anwendung zurück. Das JSON-Objekt dient in der Activity als Eingabe-Parameter der Methode *fillObjectDialog()*, die die Ausgabe der Daten für den Benutzer generiert.

4.3 Testen der Android-Anwendung

4.3.1 Zieldefinition

In diesem Abschnitt soll das Testen der mobilen Anwendung näher betrachtet werden, wobei der Fokus auf dem besonderen Verhalten der Android-Umgebung, wie die Ausrichtung und die unterschiedlichen Lebenszyklen (Pausieren, Neustart usw.), gelegt werden soll. Die Planung und Implementierung erfolgt in Anlehnung an das Tutorial „Activity Testing[6]“ der offiziellen Developer-Webseite von Android.

4.3.2 Planung

In *Tabelle 2* werden die Funktionen definiert, die als Grundlage für die zu implementierenden Test-Methoden Anwendung finden sollen.

Tabelle 2: geplante Testfälle

Nr.	Funktion	Beschreibung	Erwartetes Ergebnis
1	Initialisierung	Test, ob die Initialisierung der grafischen Elemente erfolgt	Nach Start der Anwendung sind die grafischen Objekte initialisiert
2	Geräteausrichtung	Test des Verhaltens der Applikation bei Veränderung der Geräteausrichtung	grafische Komponenten verbleiben an ihrer Position
3	Lebenszyklus	Test, ob *onPause()*, *onDestroy()* und *onResume()* den Anwendungsstatus speichern	Daten werden korrekt gespeichert und bei Neustart zurückgesichert
4	Anwendungsrechte	Testet die benötigten Anwendungsrechte	Rechte für Netzwerkzugriff und Telefonstatus sind gesetzt
5	Auswahlliste	Testet die Funktionen der Spinner	Auswahl im Spinner liefert Richtige Position und dazugehörigen Wert
6	Rückgabewerte	Testet die Rückgabewerte nach Aufruf der Methode „Zxing“	Rückgabewerte werden in entsprechendes Textfeld eingetragen
7	leere Eingabefelder	Testet die Reaktion der Anwendung auf leere Eingabefelder	Es muss eine *EmptyInputException* erzeugt werden

Die Implementierung der Testklassen erfolgt in der Entwicklungsumgebung „Eclipse“, die die Möglichkeit bereitstellt die Test-Methoden auszuführen und die Ergebnisse zu visualisieren. Im folgenden Abschnitt wird näher auf die Implementierung der geforderten Testfälle eingegangen.

[6] http://developer.android.com/resources/tutorials/testing/activity_test.html

4.3.3 Umsetzung

Nachfolgend wird die derzeitige Struktur des Test-Projektes mit Hilfe eines Klassendiagramms dargestellt und erläutert.

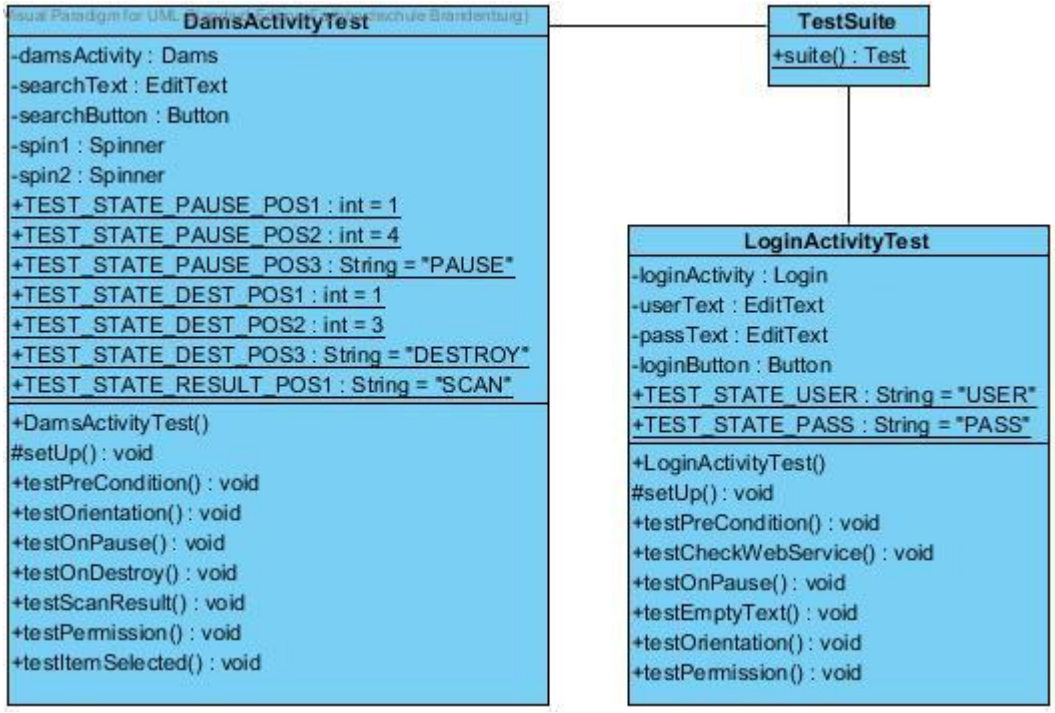

Abbildung 12: Klassendiagramm der Test-Klassen

Durch die Entwicklungsumgebung werden die Klassen *DamsActivityTest* und *LoginActivityTest* als Testklassen zu den gleichnamigen Activity-Klassen der Anwendung generiert. Zusätzlich wurde die Klasse *TestSuite* implementiert, die den Ablauf aller Testklassen, die sich im gleichen Paket befinden, steuert und somit das Starten der Tests durch automatisiertes Ablaufen vereinfacht.

Im Konstruktor der jeweiligen Test-Klassen werden Instanzen der zu testenden Klassen generiert, die die Grundlagen der folgenden Test-Methoden darstellen. In beiden Test-Klassen wird anschließend die *setUp()*-Methode aufgerufen, um die Initialisierung der Testumgebung durchzuführen. Neben den benötigten grafischen Komponenten werden zusätzlich noch Konstanten (z.B. *TEST_STATE_PAUSE _POS1*) für die Testszenarien angelegt, die in den entsprechenden Methoden als Testwerte dienen. Wie bereits erwähnt (s. *2.3.2 JUnit-Framework)* wird die *setUp()*-Methode vor jedem Aufruf einer Test-Methode ausgeführt, um die Testumgebung zurückzusetzen. Im Folgenden werden die einzelnen Methoden der Klasse *DamsActivityTest* textuell beschrieben.

- *testPreCondition()*

Diese Test-Methode wird durchgeführt, um die erfolgreiche Initialisierung durch die *setUp()*-Methode zu überprüfen. Dieses Vorgehen stellt sicher, dass die benötigten Komponenten, wie die Eingabefelder und Spinner, zur Verfügung stehen.

- *testOrientation()*

Mit Hilfe dieser Methode soll das Verhalten der mobilen Anwendung bei einer Veränderung der Geräteausrichtung getestet werden. Für die prototypische Implementierung ist die Anwendung nur im Hochformat („Portrait“) verfügbar Das führt dazu, dass eine Änderung der Ausrichtung nicht zu einer Positionsveränderung der grafischen Elemente führen darf. Diese Festlegung ist Grundlage für die folgende Implementierung.

Zu Beginn der Methode werden die X- und Y-Koordinaten der grafischen Elemente und der aktuelle Wert der Auswertung gespeichert. Mit Hilfe der Methode *setRequestedOrientation()* wird die Activity auf Querformat („Landscape“) umgestellt. Anschließend wird mit der Methode *assertNotSame()* geprüft, ob die Drehung der Ansicht erfolgt ist. Mit der Verwendung der Methode *assertEquals()* wird abschließend geprüft, ob sich die Koordinaten der grafischen Elemente in Bezug auf die Ausgangswerte verändert haben.

- *testOnPause()*

Die Methode *onPause()* der Klasse Activity wird aufgerufen, wenn die aktuelle Anwendung verlassen oder eine andere Anwendung gestartet wird. Hintergrund dafür ist, dass der aktuelle Zustand der Anwendung gespeichert wird, um ihn bei einem erneuten Aufruf zurückzusichern. Mit Hilfe der Methode *testOnPause()* soll überprüft werden ob die gespeichert Werte ordnungsgemäß beim Aufruf der Methode *onResume()* zurückgesichert werden. In dieser Methode findet die Klasse *Instrumentation* (s. *2.3.3.1 Grundlagen)* Verwendung, um den Aufruf der Methoden *onPause()* und *onResume()* zu steuern. Diese Methode ist mit *@UiThreadTest* annotiert, um Veränderungen an Werten von grafischen Elementen während der laufenden Tests durchführen zu können. Vor dem Aufruf der Methode *inst.callActivityOnPause(damsActivity)* werden die Inhalte der grafischen Elemente mit den Konstanten initialisiert

bzw. die vordefinierten Spinner-Positionen gesetzt. Nachdem die Methode *onPause()* aufgerufen wurde, werden die Spinner-Positionen verändert und der Text des Eingabefeldes gelöscht. Anschließend wird die Methode *inst.callActivityOnResume(damsActivity) ausgeführt*, um dann mit Hilfe der Methode *assertEquals()* zu prüfen, ob die nach dem Neustart der Anwendung zurückgesicherten Werte mit den Werten vor der Beendigung übereinstimmen.

- *testOnDestroy()*

Die Methode *testOnDestroy()* ist ähnlich wie die Methode *testOnPause()* implementiert. Für das Beenden und das Starten der zu testenden Aktivität wird in diesem Fall nicht die Funktionalität der Klasse *Instrumentation* verwendet. Das Beenden der Aktivität erfolgt durch den Aufruf der Methode *damsActivity.finish()*. Der Neustart der Anwendung erfolgt durch den Aufruf der Methode *getActivity()*, die durch *ActivityInstrumentationTestCase2* zur Verfügung gestellt wird.

- *testScanResult()*

Unter Verwendung der Methode *testScanResult()*, kann die Funktionalität der Methode *onActivityResult()* überprüft werden. Diese Methode wird aufgerufen, nachdem der Benutzer mit Hilfe der Anwendung „Zxing“ einen Strichcode gescannt hat. Um das erfolgreiche Einlesen überprüfen zu können, wird ein *Intent*[7] erzeugt, der die Informationen der Barcode-Anwendung speichert und als Paramater für die *onActivityResult()*-Methode bereitstellt. Der Wert des eingelesenen Strichcodes soll durch die Methode in das entsprechende Eingabefeld eingetragen werden. Die erfolgreiche Eintragung wird abschließend mit Hilfe der Methode *assertEquals()* überprüft.

- *testPermission()*

Die Verwendung von System-Ressourcen durch eine Applikation erfordert in der Android-Umgebung die Bereitstellung der entsprechenden Berechtigungen. Die *testPermission()*-Methode testet anhand der eindeutigen Namen der Berechtigungen, ob diese für die Anwendung gesetzt sind.

[7] http://developer.android.com/reference/android/content/Intent.html

- *testItemSelected()*

  Für die Auswahl entsprechender Suchkriterien stehen in der mobilen Anwendung zwei Spinner zur Verfügung. Allerdings besteht die Einschränkung, dass der zweite Spinner nur aktiv ist, wenn im ersten Spinner die erste Position ausgewählt ist. Die Überprüfung dieses Verhaltens erfolgt durch den Aufruf der Methode *damsActivity.onItemSelected()*. Eine anschließende Überprüfung durch *assertTrue()* zeigt ob das zweite Feld entsprechend den Eingaben funktionsfähig ist.

Die Test-Methoden in der Klasse *LoginActivityTest* wurden in Anlehnung an die vorherigen genannten Methoden implementiert. Aus diesem Grund werden nachfolgend nur zwei zusätzliche Methoden näher beschrieben.

- *testCheckWebService()*

  Mit Hilfe der Methode *testCheckWebService()* soll das Testen möglicher Exceptions aufgezeigt werden. (s. *Listing 1: Beispiel zur Verwendung der Methode fail()*) In diesem Test wird, der für die Überprüfung der Netzwerkverbindung benötigte, *NetworkManager* auf *null* gesetzt. Tritt bei dem Aufruf der Methode *loginActivity.startMainActivity()* eine Exception auf, führt die Methode *assertTrue()* im *catch()*-Block der Test-Methode zu einem positiven Ergebnis. Wenn die erwartete Exception nicht eintritt, wird der Test durch die Methode *fail()* im try{}-Block mit einem negativen Ergebnis beendet.

- *testEmptyText()*

  Leere Eingabefelder sollen bei der Verarbeitung zur Generierung einer *EmptyInputException* führen. Die Methode *testEmptyText()* überprüft dieses Verhalten. Aufgrund der gleichen Verfahrensweise wie bei der Methode *testCheckWebService()* wird diese Methode nicht näher erläutert.

Im letzten Kapitel erfolgt eine Auswertung der erlangten Erkenntnisse zur exemplarischen Erstellung der mobilen Anwendung und es erfolgt ein Ausblick auf mögliche Erweiterungen.

5 Abschluss

5.1 Ausblick

Bei einer weiterführenden Implementierung sind verschiedene Erweiterungen der mobilen Anwendung möglich. Denkbar sind die in *3.4 Konzeption der Anwendungsfälle* erarbeiteten Use-Cases, des Akteurs „Dams-Admin“, umzusetzen. Besonders für den Anwendungsfall „Inventur“ ist anzustreben, die Daten der Web-Anwendung mit den IST-Daten in den Rechenzentren zu vergleichen und über die Auswertung ein Änderungsprotokoll zu erstellen sowie dieses per Mail zu versenden. Weiterhin ist denkbar, mit Hilfe der mobilen Anwendung neues Inventar in die virtuellen Rechenzentren aufzunehmen und zu platzieren. Zur besseren Nutzbarkeit ist dabei auch die Realisierung einer Visualisierungsfunktion möglich, um das Auffinden gesuchter Komponenten zu vereinfachen. Für eine bessere Performance beim Einscannen von Barcodes ist in Betracht zu ziehen, dass ein vom mobilen Endgerät abgesetzter Bluetooth-Handscanner eingesetzt wird.

5.2 Fazit

In dieser Arbeit sollte grundsätzlich geprüft werden, ob eine mobile Anwendung unter Nutzung der Datenbank der vorhandenen Web-Anwendung implementierbar ist. Nach Abschluss der Arbeit, ist ein erster Prototyp der Anwendung entstanden, der in der Lage ist, mit Hilfe von diversen Suchkriterien Datensätze aus der Datenbank abzufragen und die Ergebnisse zu visualisieren. Grundlage dafür stellt ein Web-Service dar, der einen Großteil der Programm-Logik abdeckt und entsprechende Schnittstellen für die mobile Anwendung definiert. Die Implementierung des Web-Service gestaltete sich recht trivial, wobei die Komplexität der vorliegenden Datenbank die Abfrage der gesuchten Datensätze erschwerte. Das zeichnete sich vor allem bei der Umsetzung der Kabelverfolgung ab, da hier ein eigener Algorithmus implementiert werden musste.

Die Einbindung in das produktive Netzwerk muss in einem anschließenden Folge-Projekt noch untersucht werden. Die Struktur des Web-Service ermöglicht es allerdings, den Zugriff für mobile Endgeräte in die DMZ zu verlagern (s. *Abbildung 3)* und somit die grundlegende Anforderung aus der Aufgabenstellung zur erfüllen, dass kein direkter Zugriff auf das interne Server-Netzwerk gewährt werden darf. Durch die Verlagerung des Großteils der Programmlogik in den Web-

Service ist es grundsätzlich vorstellbar, die Anwendung auch auf anderen Plattenformen, wie Windows® Mobile oder Apple iOS zu portieren.

Die mobile Anwendung stellt derzeit nur beispielhafte Grundfunktionen für den Benutzer zur Verfügung. Als Grundlage für die Beurteilung der Durchführbarkeit ist die bisherige Implementierung aus Sicht des Verfassers ausreichend. Weitere Anwendungsfälle sind auf dieser Grundlage durchaus realisierbar. So konnte gezeigt werden, dass schon mit Hilfe einfacher Mittel der Zugriff auf die zugrundeliegende Datenbank ermöglicht werden kann. Nachfolgend müssen Tests der Usability und der Nutzer-Akzeptanz in der Praxis zeigen, ob die Anwendung für die Benutzer den erwarteten Nutzen bringt.

Die Implementierung der Testklassen ist ebenfalls erfolgreich verlaufen. Allerdings ist dabei die Beschränkung auf einzelne Unit-Tests erfolgt, die die speziellen Sachverhalte von Android-Anwendungen näher betrachten. Die Durchführung weiterer Tests ist in die praktische Testphase zu integrieren.

A Literaturverzeichnis

[And11] *android developers:* http://developer.android.com/guide/topics/testing/index.html (Zugriff: 28. 07. 2011).

[Bur10] BURKE, BILL: *RESTful Java with JAX-RS.* Sebastopol: O'Reilly, 2010.

[Fie00] FIELDING, ROY THOMAS: *www.ics.uci.edu.* 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (Zugriff: 20. 06. 2011).

[Gra2005] GRAHAM, STEVE; *Building Web Services with Java.* Indianapolis, Sams Publishing, 2005.

[HT03] HUNT, ANDREW; THOMAS, DAVID: *Der Pragmatische Programmierer.* München: Hanser, 2003.

[Jso11] www.json.org/xml.html (Zugriff: 23. 07. 2011).

[Jun11] JUNGINGER, MARKUS: „Historie der Android-Versionen.“ *android360*, 01. 06. 2011.

[Krä11] KRÄMER, WALTER: *http://www.speedikon-dams.de.* http://www.speedikon-dams.de/download/produktinformation.pdf (Zugriff: 20. 07. 2011).

[MH04] MASSOL, VINCENT; HUSTED, TED: *JUnit in Action.* Greenwich: Manning, 2004.

[RS05] RAINSBERGER, J.B.; STIRLING, SCOTT: *JUnit Recipes.* Greenwich: Manning, 2005.

[Sch11] SCHUMANN, JENS: *heise online.* 23. 05. 2011. http://www.heise.de/developer/artikel/Webservices-mit-Java-EE-6-JAX-WS-und-RESTful-Services-1247464.html (Zugriff: 20. 06. 2011).

[Wor11] *World Wide Web Consortium.* http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html (Zugriff: 16. 06. 2011).

B Abkürzungsverzeichnis

API
Application Programming Interface

CPU
Central Processing Unit

DBMS
Datenbank Management System

DMZ
Demilitarized Zone

EAN
European Article Number

HTTP
Hyper Text Transfer Protocol

IP
Internet Protocol

ISBN
International Standard Book Number

JAX-RS
Java™ API for RESTful Web-Services

JPA
Java™ Persistence API

JSON
JavaScript Object Notation

MAC
Media Access Control

MD5
Message-Digest Algorithm 5

PC
Personal Computer

PDA
Personal Digital Assitant

REST
Representational State Transfer

SQL
Structured Query Language

TCP
Transmission Control Protocol

UDP
User Datagram Protocol

URI
Uniform Resource Identifier

WLAN
Wireless Local Area Network

XML
Extensible Markup Language

C Abbildungsverzeichnis

D Listings

E Tabellenverzeichnis

F Anhang (CD)

Auf der CD befinden sich folgende Dokumente und Dateien:

- Bachelorarbeit
- Projektdateien
    - Web-Service (DAMS02)
    - Mobile Applikation (Dams)
    - Test-Klassen (DamsTest)
- Online-Quellen
- Abschlussposter