Inhaltsverzeichnis

1. Einleitung 4

1.1. Motivation 4

1.2. Aufgabenstellung 5

1.3. Aufbau und Methodik 5

2. Theoretische Vorbetrachtung **Fehler! Textmarke nicht definiert.**

2.1. Zielplattform Android 7

2.2. Web-Service 7

2.3. Software-Tests 13

2.3.1. Einführung 13

2.3.2. Unit-Test 14

2.3.3. JUnit-Framework 16

2.3.4. Testen in Android 19

2.3.4.1. Grundlagen 19

2.3.4.2. Activity Testing 21

2.3.4.3. Service Testing 23

2.3.4.4. Content Provider Testing 24

3. Umsetzung 25

3.1. Analyse „Data Asset Management Solution“ 25

3.1.1. Funktionen der Webanwendung 25

3.1.2. Verwendung in der Projektumgebung 27

3.1.3. Konzeption der Anwendungsfälle 27

3.1.4. Datenbank 29

3.2. Realisierung des Web-Service 32

3.2.1. Zieldefinition 32

3.2.2. Planung 32

3.2.3. Umsetzung 33

3.3. Realisierung der Android-Anwendung 36

3.3.1. Zieldefinition 36

3.3.2. Planung 36

3.3.3. Umsetzung 38

3.4. Testen der Android-Anwendung 45

3.4.1. Zieldefinition 45

3.4.2. Umsetzung 45

4. Abschluss 50

4.1. Erweiterungsmöglichkeiten 50

4.2. Fazit 50

[Abbildung 1: Aufbau der Arbeit 6](#_Toc300596282)

[Abbildung 2: Mögliche Anwendungsfälle der mobilen Anwendung 27](#_Toc300596283)

[Abbildung 3: Use-Case-Beschreibung "Suche starten" 29](#_Toc300596284)

[Abbildung 4: Ausschnitt der Datenbank 30](#_Toc300596285)

[Abbildung 5: Klassendiagramm Web-Service 33](#_Toc300596286)

[Abbildung 6: Screenshots der mobilen Anwendung 37](file:///D:\FHB\Abschlussarbeit\Schriftliche_Arbeit\Arbeit_2.docx#_Toc300596287)

[Abbildung 7: Klassendiagramm der mobilden Anwendung 38](#_Toc300596288)

[Abbildung 8: Sequenzdiagramm "Login" und "Suche starten" 42](#_Toc300596289)

[Abbildung 9: KLassendiagramm der Test-Klassen 45](#_Toc300596290)

# Einleitung

## Motivation

Die Verwaltung zweier Rechenzentren in einem täglich produzierenden Zeitungsverlag, wie bei der Verlag Der Tagesspiegel GmbH, ist sehr komplex, da dafür die unterschiedlichsten Inventardaten der verbauten Komponenten erfasst werden müssen. Naturgemäß ist dabei auch die Fehlersuche während der laufenden Produktion sehr zeitkritisch, d.h. die fehlerhaften Geräte müssen schnell ausfindig gemacht werden können. Um diese Aufgaben bewältigen zu können 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 der 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 zum einen darin, dass die Daten lokal auf dem PDA vorgehalten werden und somit nicht mit dem Stand der Datenbank übereinstimmen müssen. Das erfordert eine ständige Synchronisierung zwischen Datenbank und PDA. 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 möglicherweise 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 Echtzeit-Zugriff 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 eingebaute Kamera in Verbindung mit einer portablen Anwendung, können unter Umständen zu einer Akzeptanzsteigerung der vorhandenen Web-Anwendung führen.

## Aufgabenstellung

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 Benutzeroberfläche zur Verfügung stellen, die nach einer erfolgreichen Anmeldung des Nutzers die Suche und Abfrage von Inventardaten mit Hilfe diverser Suchkriterien ermöglicht und eine entsprechende Ausgabe generiert. Mit Hilfe der entstehenden Anwendung soll zudem überprüft werden, ob der Einsatz von mobilen Endgeräten im administrativen Bereich eine Sinnhaftigkeit und Durchführbarkeit besitzt.

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 an das vorhandene WLAN-Netzwerk des Verlages. 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 Umsetzung in Android stellt somit nur eine Test-Plattform dar.

## Aufbau und Methodik

Die vorliegende Arbeit ist in vier Teilabschnitte gegliedert (siehe Abbildung 1: Aufbau der Arbeit). Nach einer Einführung mit der Beschreibung der Problem- und Aufgabenstellung folgt in Kapitel zwei, die theoretische Vorbetrachtung des Projektes. Dabei wird auf die theoretischen Grundlagen und die Begründung der Nutzung eins selbst zu implementierenden Web-Service eingegangen. Weiterhin folgt ein kurzer Überblick zum Thema Android Betriebssystem. Als Abschluss der theoretischen Betrachtung soll eine kurze Einführung in diverse Softwaretests gegeben werden. 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 : Aufbau der Arbeit

Das dritte Kapitel beschreibt die Umsetzung der einzelnen Komponenten. Im ersten Teilabschnitt werden dafür die Web-Anwendung „speedikon DAMS“ und die zugrundeliegende Datenbank, näher betrachtet und die Funktionalitäten der Anwendung beschrieben. Aus der Betrachtung von „speedikon DAMS“ heraus werden mögliche Anwendungsfälle für mobile Anwendung abgeleitet, die zum Teil zur Implementierung herangezogen werden. Im Anschluss folgen die Kapitel zur Umsetzung des Web-Service, der mobilen Anwendung und der Realisierung der Unit-Tests. Als Abschluss der Arbeit wird in Kapitel vier ein Fazit gezogen, dass die Ergebnisse der Betrachtungen kurz zusammengefasst. Abschließend werden diverse Erweiterungsmöglichkeiten der mobilen Anwendung für eine eventuelle vollständige Implementierung aufgezeigt.

# Umsetzung

## Zielplattform Android

Für die praktische Realisierung der Projektaufgabe ist die Verwendung von Android als Betriebssystem vorgesehen. Nachfolgend soll ein kurzer Einblick in die Vielfalt der Android-Umgebung geschaffen werden.

## Web-Service

Wie in der Einleitung beschrieben, stellt die zugrundeliegende Web-Anwendung keine Schnittstellen für den Zugriff auf die vorhandenen Daten zur Verfügung, so dass ein entsprechender Web-Service implementiert werden soll. In diesem Abschnitt sollen die Grundlagen zum Thema Web Service erläutert werden.

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. Web-Anwendungen wie Amazon, Ebay und Google bieten Schnittstellen zur Nutzung der eigenen Dienste zur Einbettung in anderen Anwendungen an.

Für den Austausch und die Manipulation von Daten über definierte Schnittstellen hat sich die Architektur REST (Representational State Transfer) etabliert [HEI10]. Die Grundlagen hat Roy Fielding im Jahr 2000 in seiner Doktorarbeit gelegt. [FIE00]. Nachfolgend sollen die Eigenschaften von REST aufgezeigt werden.

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 Ergebnisse von Datenbankabfragen vorliegen, die in unterschiedlichen Formaten wie XML3 oder JSON4 repräsentiert zur Verfügung gestellt werden können. Für den Aufruf wird jede der Ressourcen mit einem eindeutigen „Uniform Resource Identifier“ (URI) adressiert. Die Idee bei REST liegt in der einfache Manipulation der angesprochen Ressourcen. Dabei sollen einfache Schnittstellen definiert werden, um die so genannten CRUD-Methoden auf die gewünschten Ressourcen anwenden zu können. CRUD steht für folgendes:

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

Als Transportprotokoll nutzt REST2 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.[Ker09] HTTP beschränkt sich dabei auf eine geringe Anzahl von Methoden, die hier kurz aufgezeigt werden sollen:

* GET

Die Methode GET stellt den lesenden Zugriff auf Ressourcen zur Verfügung. Die Methode verhält sich dabei idempotent und ist sicher bzw. gefahrlos. Idempotent bedeutet in diesem Fall, dass ein lesender Zugriff die gewünschte Ressource unverändert lässt und somit jeder weitere Zugriff auf die Ressource das gleiche Ergebnis liefert. Sicher im Zusammenhang mit der Methode GET bedeutet, dass der Aufruf einer Ressource durch einen Benutzer keinen Seiteneffekt hervorruft, d.h. dass die Anfrage keinen vom Benutzer erwarteten Einfluss auf andere Ressourcen oder den Betrieb des Servers haben darf. [W3C][Bur10]

* PUT

Die Methode PUT ist eine schreibende Methode. Bei der Anfrage an den Server, wird die in der http-Nachricht übertragen Ressource an der eindeutigen Adresse (URI) gespeichert, die in den Meta-Daten des Paketes angegeben ist. Sollte die angegebene URI nicht vorhanden sein, wird sie durch den Server angelegt. Die Methode PUT ist idempotent, d.h. dass die mehrmalige speichernde Anfrage der gleichen Ressource dasselbe Ergebnis herbeiführt. [Bur10]

* DELETE

Die Methode DELETE wird benutzt um Ressourcen auf dem Server zu löschen. Die Methode ist ebenfalls idempotent, da das mehrmalige Löschen einer Ressource das gleiche Ergebnis liefert. [Bur10]

* POST

Die Methode POST ist sehr flexibel einsetzbar. So ist es möglich in den entsprechenden Anfragen bzw. Antworten Daten zu senden oder zu empfangen. Weiterhin können mit der Methode auch die Funktionalitäten von anderen Methoden wie GET und PUT realisiert werden. Allerdings ist PUT weder idempotent noch sicher, weil die in den Anfragen übertragenen Meta-Daten bzw. Parameter ständig veränderbar sind. Zudem bietet PUT weitere Funktionen an, wie das Annotieren von bekannten Ressourcen. [Bur10] [W3C]

* HEAD

Die Methode HEAD stellt die gleiche Funktionalität wie die Methode GET zur Verfügung. Der Unterschied besteht darin, dass der Server bei einer HEAD-Anfrage nicht zwingende den Inhalt der aufgerufenen Ressource zurücksendet, sondern nur die Meta-Daten die die Ressource betreffen.[W3C]

* OPTIONS

Die Methode OPTIONS ermöglicht den Aufruf von Optionen für die Kommunikation zwischen Server und Nutzer und deren Auswertung, ohne die Ressource selbst anzufordern. [Bur10]

* TRACE

Die Methode TRACE stellt Funktionalitäten zur Verfügung mit denen der Nutzer zu Testzwecken Anfragen an den Server schickt und der Server die verarbeitet Anfrage zurücksendet. Mit dieser Methode kann der Nutzer die Rückantwort auswerten und die gewonnenen Informationen zur Fehlerbehebung oder Diagnose nutzen. [W3C]

* CONNECT

Die letzte zu betrachtende Methode ist CONNECT. Diese Methode steht für die Nutzung von Proxy-Eigenschaften zur Verfügung, um dynamisch auf gesicherten bzw. getunnelten Netzwerkverkehr umzuschalten. [W3C]

Neben dem hohen Bekanntheitsgrad bietet HTTP noch weitere Vorteile. In vielen Programmiersprachen stehen Klassen für die Programmierung von Clients zur Verfügung. 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.

Eine weitere grundlege Eigenschaft der REST-Architektur ist die zustandslose Kommunikation. Zustandslos bedeutet in diesem Zusammenhang, 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 das Framework JAX-RS zur Verfügung. JAX-RS stellt Funktionalitäten bereit um „RESTful“-Webservices zu implementieren. Unteranderem bietet es die Möglichkeit Annotationen zu verwenden, um Meta-Daten die für den Webservice benötigt werden einbinden zu können. Im Folgenden sollen kurz die wichtigsten Annotationen aufgezeigt werden.

@Path

Definiert den Pfad mit der die zur Verfügung gestellte Klasse aufgerufen werden kann und bildet so den Knotenpunkt („root resource“) für die mit @Path annotierten Methoden in der Klasse.

@Path(”/test”)

public class TestKlasse{

@Path(/methode1)

public String testMethode1(){

…

}

}

Die im Web-Service aufzurufende URI wäre dann

*http://server:port/webprojekt/test/method*

Für die Einbindung von Meta-Daten der aufgerufenen Methoden gibt es weitere Annotationen:

@GET

Bindet die GET-Methode an die in der HTTP-Anfrage definierte Methode. Diese Annotation ist auch für die HTTP-Methoden PUT, POST, DELETE und HEAD möglich.

@Produces

Über diese Annotation wird der erwartete Rückgabetyp der aufgerufenen Methode festgelegt.

@PathParam

Die Annotation @PathParam dient dazu, die Eingabeparameter einer Methode mit den Werten aus der HTTP-Anfrage zu initialisieren. Der Parameter wird dann in der @Path Annotation der Methode mit geschweiften Klammern angegeben.

Beispiel:

@Path(“/methode2/{parameter}”)

public String testMethode2(@PathParam(“parameter”) String eingabe){

…

}

@FormParam

Die Annotation @FormParam bietet die Möglichkeit Parameter in Form von Name-Werte-Paaren in einer HTTP-Anfrage zu übertragen. Das Einfügen der möglichen Parameter in die HTTP-Anfrage liegt in Verantwortung des Clients.

@Path(“/methode3/”)

public String testMethode3(@FormParam(“parameter”) String eingabe){

…

}

## Software-Tests

### Einführung

Das Testen ist ein wichtiger 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 aufgeführt werden:

* Unit-Test

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

* 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

Bei diesen Tests ist es wichtig herauszufinden, ob die erstellte Anwendung den gestellten Ansprüchen des Auftraggebers entspricht. Zudem muss geklärt werden, ob die Anforderung des Auftraggebers die Wünsche der Anwender erfüllen.

* Ressourcenverbrauch, Fehlersituation und Wiederherstellung

Neben der korrekten Funktionsweise des erstellten Quelltextes ist auch ist auch die Lauffähigkeit auf dem Zielsystemen zu überprüfen. Die verfügbaren Ressourcen des Zielsystems sind zudem begrenzt. Dazu zählen unteranderem der Speicher, Festplattenplatz und die Leistungsfähigkeit der CPU. 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 Anforderung der Benutzer 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.

Nachfolgend sollen die Betrachtungen der Unit-Tests und die Umsetzung für Java mit JUnit erfolgen.

### Unit-Test

Wie bereits beschrieben stellen Unit-Tests Funktionen bereit, um Module eines Programmes, ohne die Einwirkung anderer Module zu testen. Bei der Nutzung von Unit-Tests sollen einige Grundsätze beachtet werden, die nachfolgend kurz aufgezeigt werden sollen.

* Automatisch

Der Aufruf der Testklassen und die Auswertung der Ergebnisse soll automatisch erfolgen. Der Hintergrund dafür besteht in der Wiederverwendbarkeit der Tests, die sofort aufzeigen sollen falls ein Test fehlschlägt. Der manuelle Aufruf birgt die Gefahr, dass Tests vergessen werden bzw. die Auswertung nicht erfolgt. Zudem sollten Tests verwendet werden, die ohne die manuelle Eingabe von Testparametern auskommt, um den automatischen Ablauf zu gewährleisten. Um den automatischen Ablauf weiterhin zu gewährleisten sollen für die Tests Mock-Objekte und isolierte Ablaufumgebung (siehe2.3.4.1)verwendet werden.

* Sorgfalt

Die zu implementierende Testklassen sollen alle möglichen auftretenden Fehlerfälle durchlaufen und eine möglichst große Code-Abdeckung realisieren. Bei großen Software-Projekten kann das Testen der gesamten Funktionalitäten langwierig und unübersichtlich werden. Abhängig von den gewünschten Projektzielen, kann es ausreichen die Tests auf bestimmte Szenarien (falsche Eingaben, Null-Pointer, falsche Datentypen und Rückgabenwerte, o.ä.) zu beschränken.

* Wiederholbar

Die durchzuführenden Tests müssen zu einem Zeitpunkt, in jeder Reihenfolge in der sie ablaufen, die gleichen Ergebnisse liefern. Für diesen Ablauf ist die Unabhängigkeit der Testklassen untereinander erforderlich. In größeren Projekten, in denen mehrere Entwickler tätig sind, ist zu beachten, dass die Tests der einzelnen Entwickler sich gegenseitig nicht beeinträchtigen dürfen.

* Unabhängigkeit

Testklassen dürfen keine Abhängigkeiten zuvor ablaufender Tests besitzen. Um diesen Aspekt gewährleisten zu können, sollen die setUp() und tearDown()-Methoden (siehe 2.3.3)genutzt werden, um die Testumgebung nach jeder Testklasse zurückzusetzen.

* Professionell

Die Implementierung der Testklassen ist mit gleicher Präzision durchzuführen, wie bei dem Produkt-Code der getestet werden soll. Die Testklassen sind den zu testenden Methoden entsprechend anzupassen, so dass die Funktionalitäten methodenspezifisch überprüft werden. Weiterhin sind die Grundsätze der Software-Entwicklung, wie „DRY- Don’t repeat yourself“ einzuhalten.

### JUnit-Framework

Das JUnit-Framework ist aus dem von Kent Beck entwickelten SUnit-Framework für die Programmiersprache Smalltalk hervorgegangen. [RAI05] Das JUnit-Framework stellt Methoden zur Verfügung um Unit-Tests durchzuführen. Unter anderem zählen dazu Methoden mit denen die Testumgebung vorbereitet werden kann, die die Ergebnisse visualisieren und die getesteten Objekte freizugeben. 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 Unittest 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.

Neben den Klassen zur Vor- und Nachbereitung der Testumgebung stehen im JUnit-Framework weitere Methoden zur Verfügung, um Behauptungen aufzustellen und zu überprüfen, ob diese erfüllt werden. Die Methoden stehen in der Klasse Assert zur Verfügung, die durch die Klasse TestCase erweitert wird. 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 Nachricht kann optional angegeben werden und 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 nicht 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 soll die Funktionsweise kurz verdeutlichen:

public void testException() {

try{

methode4(null);

fail(“Die Methode sollte eine Exception werfen!”);

} catch (Exception e){

assertTrue(true);

}

}

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 geworfen, wird die Methode fail() mit der entsprechenden Nachricht aufgerufen und somit erzwungen, dass der Test nicht erfolgreich ist.

### Testen in Android

Im folgenden Abschnitt sollen die Möglichkeiten der Software-Tests von Android-Anwendungen näher beschrieben werden. Als Grundlage für die Ausführung wird das offizielle Entwickler-Forum von Android genutzt.

#### 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 Klasse 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 seine, die eine Datenbankveränderung hervorrufen würde. Da die Datensätze durch die durchgeführten Tests nicht verändert werden dürfen, wird ein Mock-Objekt erzeugt, welches isoliert von der realen Datenbank auf den Methodenaufruf reagiert.

* 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 Context verarbeitet.

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

#### Activity Testing

Die Klasse Activity stellt die Grundlage für die Android-Anwendung mit Benutzeroberflächen dar, mit denen der Benutzer interagieren kann. Eine Anwendung kann aus mehreren Activities bestehen, die sich auch gegenseitig starten können, wenn die entsprechenden Berechtigungen bestehen. Für das Testen von Activities, steht die bereits erwähnte Instrumentation API bereit. Die API beinhaltet die Klasse InstrumentationTestCase, die die Oberklasse für die zu implementierenden. Folgende Hauptunktionen sollen durch die Klasse abgedeckt werden:

* Kontrolle des Lebenszyklus

Android-Anwendung durchlaufen verschiedene Lebenszyklen. Dazu zählen unteranderem das Starten der Anwendung, das Pausieren, wenn andere Anwendungen aufgerufen werden, das Zurückkehren von anderen Anwendungen und das Beenden der Anwendung. Für jeden dieser Fälle stehen in der 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 Testklassen stehen drei Klassen zur Verfügung, die die Klasse InstrumentationTestCase implementieren. Diese Klassen unterscheiden sich in 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 durchzuführen Tests wird eine Instanz der zu testenden Anwendung in der normalen System-Umgebung generiert. Für den Aufruf anderer Activities können Mock Intents eingesetzt werden. Bei Verwendung dieser Oberklasse können allerdings keine Mock-Objekte 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-Context möglich. Die Nutzung von Mock-Objekten anderer Activities ist nicht möglich.

* SingleLaunchActivityTestCase

Die Klasse SingleLaunchActivityTestCase wird für Testumgebung 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.

#### 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 der Service gestartet wurde. 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 TestCase 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.

#### 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 der ContentResolver erzeugt, der für die bereitgestellten Daten entgegen nimmt. Abschließend wird ein Objekt der Klasse ContentProvider erzeugt, dass durch die vorherige Initialisierung in einer isolierten Testumgebung abläuft.

# Umsetzung

## Analyse „Data Asset Management Solution“

Im folgenden Teilabschnitt soll die zugrundeliegende Anwendung „speedikon DAMS“ näher betrachtet werden. Dabei soll kurz auf die Funktionalitäten der Web-Anwendung und die derzeitige Verwendung im Verlag eingegangen werden. Das Hauptaugenmerk soll auf die Datenbankstruktur der Anwendung gelegt werden. Zusätzlich sollen Anwendungsfälle für eine portable Version aufgezeigt werden.

### Funktionen der Webanwendung

Die Web-Anwendung „Datacenter Asset Management Solutions“ (DAMS) der Firma speedikon bietet diverse Möglichkeiten zur Verwaltung und Dokumentation von Bestandsdaten in Rechenzentren an. Nachfolgend sollen kurz einige Möglichkeiten aufgezeigt werden.

* Verwaltung der Bestandsdaten

Die Hauptaufgabe die mit DAMS realisiert werden kann, ist die Verwaltung von Bestandsdaten. Zu diesen Bestandsdaten können unteranderem Server, Netzwerkgeräte, Peripherie-Geräte und ähnliche Komponenten. gehören. Für jedes zu pflegende Gerät müssen diverse Informationen vorgehalten werden. Neben technischen Informationen wie die Stromaufnahme und Wärmeabgabe eines Gerätes können auch Netzwerkdaten, wie IP- und MAC-Adresse registriert werden. Weiterhin besteht die Möglichkeit, kaufmännische Daten wie Inventarnummern, Seriennummern oder ähnliche Informationen zu speichern. Durch die große Vielfalt der gerätespezifischen Daten die gespeichert werden, 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 festzuhalten- Einerseits besteht so die Möglichkeit, die verbauten Geräte zeitnah mit Hilfe einer Suchfunktion wiederzufinden. Andererseits kann durch die lückenlose Dokumentation die Planung neuer Geräte unterstützt werden, um freie Standorte für zu verbauende Komponenten ausfindig zu machen. Ergänzend zur eigentlichen Dokumentation kann eine 2D und 3D Visualisierung der Rechenzentren erfolgen. Das sorgt für einen hohen Widererkennungswert der Rechenzentren und unterstützt zusätzlich die Suche von Komponenten.

* Kabeldokumentation

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

* Reports

Mit Hilfe von DAMS ist es möglich, diverse Auswertungen in Form von Reporten zu generieren. Für die langfristige Planung der Rechenzentren können diese Auswertungen genutzt werden, um einen Überblick über freie Ressourcen und die aktuelle Auslastung der Rechenzentren zu bekommen.

### Verwendung in der Projektumgebung

Derzeit wird DAMS im Verlag genutzt, um die Dokumentation der Bestandsdaten durchzuführen. Dabei werden die in 3.1.1 beschriebenen Gerätedaten und Kabelwege aufgenommen. Zur Unterstützung der Kabeldokumentation sind an sämtlichen Netzwerk-Kabeln eindeutige Nummern mit entsprechendem Strichcode angebracht worden. Für die Strom-Kabel 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 Strom-Kabel wird aufgrund der geringen Priorität zu einem späteren Zeitpunkt durchgeführt.

### Konzeption der Anwendungsfälle

Nachfolgend soll mit Hilfe eines Use-Case-Diagramms (siehe Abbildung 2) mögliche Anwendungsfälle für eine mobile Anwendung aufzuzeigen.



Abbildung : Mögliche Anwendungsfälle der mobilen Anwendung

Für die mobile Anwendung werden zwei Akteure mit verschiedenen Aufgabengebieten unterschieden. Der Akteur „Benutzer“ stellt einen Anwender mit eingeschränkten Benutzerrechten dar, der nur die Suche 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 dem Anwender ermöglicht werden, den Suchparameter-Typ 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“ 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 die in der Abbildung 1 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 des Akteurs „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 des Prototyps zu schaffen.

|  |  |
| --- | --- |
| Name | Suche starten |
| Kurzbeschreibung | Ermöglicht den Abruf von Bestandsdaten mit Hilfe verschiedener Such-Parametern |
| Akteure | Benutzer, DAMS-Admin |
| Auslöser | Aufruf der Suche durch einen Akteur |
| Vorbedingung | Die Netzwerk-Verbindung zur Datenbank ist vorhanden. |
| Ergebnis | Anzeige der gewünschten Fahrerdaten |
| Nachbedingung | Es kann eine erneute Suche durchgeführt werden. |
| Essenzieller Ablauf | Aufruf der Such-Funktion  Auswahl der gewünschten Suche (Geräte- oder Kabelsuche)  Auswahl des Such-Parameter-Typs  Eingabe des Such-Begriffs (manuell oder einscannen)  Ausgabe der Bestandsdaten |

Abbildung 3: Use-Case-Beschreibung "Suche starten"

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

### Datenbank

In diesem Teilabschnitt soll kurz der Aufbau der zugrunde liegenden 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 ER-Diagramm zu skizziert in dem die wichtigsten Tabellen, einschließlich ihrer Beziehungen, zu sehen sind.



Abbildung : Ausschnitt der Datenbank

Die drei Tabellen stellen die benötigten Datensätze für den Use-Case „Suche starten“ bereit. Die Tabelle „dbo.sap\_objects“ stellt die zentrale Instanz der Web-Anwendung dar. In dieser Tabelle werden alle Objekte mit ihren entsprechenden Attributen gespeichert. Die Analyse der Tabelle ist nicht trivial, da die die Spalten 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, die im originalen Datenbank-Schema einen eindeutigen Namen trägt. Die Tabelle „dbo.sap\_net\_interfaces“ speichert die Daten sämtlicher Netzwerk-Schnittstellen, 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 Gerät und der verbauten Netzwerk-Schnittstelle 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 den Use-Case „Kabel suchen“ und „Verkabelungswege verfolgen“ realisieren zu können. Neben den fehlenden Schlüsselbeziehungen im 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 berücksichtigt werden.

## Realisierung des Web-Service

### 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-Services geschaffen werden. Ziel soll es sein, die Login- und Such-Anfragen der Benutzer zu verarbeiten und die Ergebnisse in Form von JSON-Objekten zurückliefern.

### Planung

Für den Zugriff auf die Datenbank der Web-Anwendung soll die Java-Persistence-API (JPA) verwendet werden. Für die Generierung der Persistence-Objekte stellt das JPA-Framework Funktionalitäten bereit, um diese Objekte durch die Entwicklungs-Umgebung zu generieren. Weiterhin bietet JPA die Möglichkeit, Datenbank-Anfragen an die Datenbank zu schicken und die Ergebnisse entgegen zunehmen. Die Nutzung von JPA erleichtert somit den Zugriff auf die Datenbank, weil die Steuerung der Verbindungen zur Datenbank und das OR-Mapping durch das Framework realisiert werden. Die Ergebnisse der Datenbank-Abfragen sollen in Form von JSON-Objekten für die mobile Anwendung bereitgestellt werden. JSON-Objekte sind einfach zu implementierende Dateiformate, die sowohl maschinen- als auch menschenlesbar sind. Die Definition der Schnittstellen wird durch das Framework Jersey realisiert, das durch die Nutzung von Annotationen (siehe *Web-Service*) die zu implementierenden Methoden für andere Anwendungen bereitstellt. Als Applikations-Server wird ein Apache Tomcat in der Version 7 zum Einsatz kommen.

### Umsetzung

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

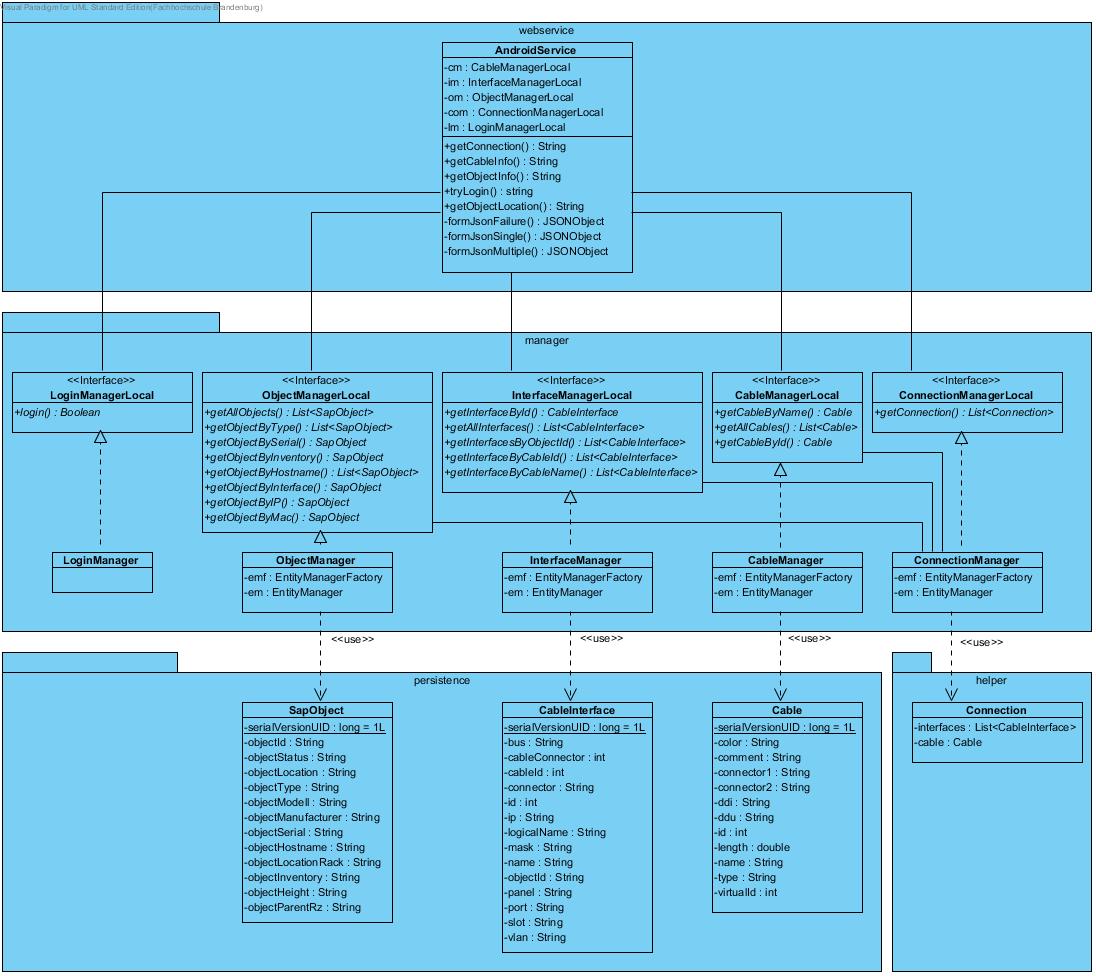


Abbildung : Klassendiagramm Web-Service

Die Struktur des Web-Service ist in die vier Pakete „persistence“, „manager“, „webservice“ und „helper“ unterteilt. Die durch die Entwicklungsumgebung generierten Persistenz-Klassen befinden sich im gleichnamigen „persistence“-Paket und repräsentieren die Abbildung der relationalen Tabellen der zugrundeliegenden Datenbank. Das Paket „helper“ beinhaltete die Klasse „Connection“, die als Hilfsklasse für das Auffinden von Netzwerk-Verbindungen, bestehend aus mehreren Netzwerk-Kabeln implementiert wurde. Die Idee besteht darin, dass ein Netzwerk-Kabel vom Typ „Cable“ mit höchstens zwei „CableInterfaces“ verbunden sein kann. Die „CableInterfaces“ können über das Attribut „objectId“ eindeutig dem dazugehörigen „SapObject“ zugordnet werden. Die Klasse „ConnectionManager“ stellt über das Interface „ConnectionManagerLocal“ die Funktionen bereit, um Netzwerk-Verbindungen zwischen zwei Geräte ausfindig zu machen. Ergebnis dieser Suche ist eine Liste, in der jedes Element ein Objekt vom Typ „Connection“ ist.

Neben dem Interface „ConnectionManagerLocal“ und der implementierenden Klasse „ConnectionManager“ stehen weitere Klassen und Interfaces für den Zugriff auf die Persistenz-Objekte zur Verfügung. Die Klassen „ObjectManager“, „InterfaceManager“ und „CableManager“ implementieren, die namentlich mit „Local“ erweiterten Interfaces. Die Einbindung der Persistenz-Objekte erfolgt dabei über die „EntityManager“, die durch die „Java Persistence API“ bereitgestellt werden. Die passenden Interfaces stellen anderen Klassen die Methoden bereit, um die gewünschten Objekte anhand bestimmter Such-Kriterien aufzurufen. Neben dem lesenden Zugriff stellen die Interfaces normalerweise weitere Methoden bereit für die Anwendung der sogenannten CRUD-Methoden (siehe 2.2) bereit. Aufgrund der Einschränkungen in diesem Projekt, wurde auf diese Implementierung verzichtet.

Zusätzlich befindet sich in dem Paket „manager“ noch die Klasse „LoginManager“ mit dem passenden Interface „LoginManagerLocal“. 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 Android-Service an die Klasse „LoginManager“ über das Interface „LoginManagerLocal“ weitergegeben. Die Klasse wertet die Einträge der zuvor gefilterten Benutzer-Einträge aus und gibt den entsprechenden Wahrheitswert eines positiven oder negativen Anmelde-Versuchs zurück.

Die Klasse „AndroidService“ stellt den zentralen Einstiegspunkt für externe Anwendungen auf den Web-Service und die implementierten Methoden bereit. Die entsprechenden Methoden werden mit Hilfe entsprechender Annotationen bereitgestellt. Die benötigten Annotationen werden durch das Framework Jersey zur Verfügung gestellt. Jersey stellt im Java-Umfeld eine Referenz-Implementierung für RESTful Web-Services dar. [QUELLE] Neben den Methoden zur Realisierung der gewünschten Use-Cases werden durch die Klasse „AndroidService“ auch private Methoden für die Umformung der Datensätze in, die zur Übertragung via HTTP benötigten, JSON-Objekte implementiert.

## Realisierung der Android-Anwendung

### Zieldefinition

Das Ziel der zu entwickelnden Android-Anwendung ist es, die vorhandenen Daten der Web-Anwendung „speedikon DAMS“ abzurufen und dem Benutzer auf geeignete Weise bereitzustellen. Das Ziel-System der Entwicklung ist das Samsung Galaxy Tab mit dem Betriebssystem Android in der Version 2.2. Die Anwendung soll möglichst die Anwendungsfälle des Akteurs „Benutzer“ abbilden. Als Ergänzung zu der manuellen Eingabe von Suchparametern soll die Einbindung der Strichcode-Anwendung „Zxing“ realisiert werden.

### Planung

Im Folgenden soll die Planung der mobilen Anwendung dargelegt werden. Die prototypische Implementierung soll die Möglichkeiten prüfen, ob die Anwendung auf einem tragbaren mobilen Gerät Sinn gemacht. Aus diesem Grund soll das Layout der Anwendung funktional gestaltet werden.



Abbildung : Screenshots der mobilen Anwendung

Die Screenshots in **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigen die Oberfläche der mobilen Anwendung. Der erste Ausschnitt zeigt den Anmelde-Bildschirm der mobilen Anwendung. Ein negativer Anmelde-Versuch oder leere Eingabe-Felder werden dem Benutzer mit einem „Toast“ angezeigt. Beim Start der Anwendung wird eine Überprüfung durchgeführt, ob die Datenbank erreichbar ist. Das Ergebnis dieser Überprüfung wird dem Benutzer ebenfalls angezeigt. Bei einer negativen Überprüfung der Netzwerk-Verbindung zur Datenbank wird die Anwendung, mit Aufforderung zur Herstellung einer Verbindung, beendet. Ist der Login durch den Benutzer erfolgreich wird die im mittleren Bild gezeigte Ansicht gestartet. Mit Hilfe der zwei Auswahllisten können die gewünschten Such-Kriterien ausgewählt werden. In das Texteingabe-Feld können die Suchparameter manuell eingetragen werden. Durch langes drücken des Texteingabe-Feldes soll der Barcode-Scanner Zxing gestartet werden. Bei einem erfolgreichen Scan-Versuch wird der entsprechende Wert in das Eingabe-Feld eingetragen und die Anwendung Zxing zuvor wieder beendet. Über den Button „Suche“ können die gewünschten Informationen abgerufen werden. Die Ausgabe erfolgt wie im dritten Bildausschnitt beispielhaft gezeigt. Die negative Suche nach gewünschten Objekten wird dem Benutzer mit einer Nachricht angezeigt.

### Umsetzung

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

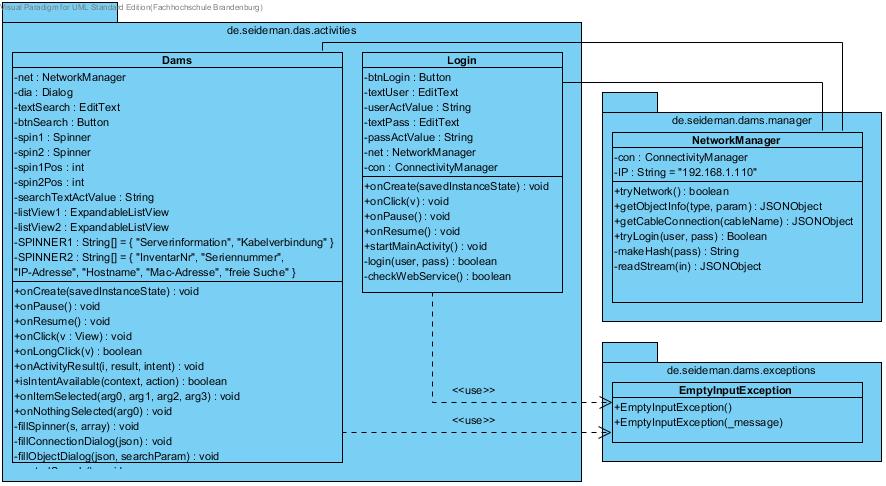


Abbildung : Klassendiagramm der mobilden Anwendung

Die Klasse *Dams* stellt die Hauptfunktionalitäten der mobilen Anwendung unter Verwendung der Klasse *NetworkManager* bereit. 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 WebService (*checkWebService()*) gestestet. Die Activity *Dams* ist hauptsächlich für die Steuerung und die grafische Ausgabe der gewünschten Such-Ergebnisse zuständig. Im Für die Realisierung der Oberfläche sind daher einige Elemente notwendig, die bei dem Starten der Anwendung initialisiert werden. Die grafischen Elemente werden dafür in einer Konfigurationsdatei mit den entsprechenden Parametern angelegt. Zur Laufzeit werden die einzelnen Elemente über die Methode *findViewById(R.id.name)* lokalisiert, um entsprechende Objekte zu initialisieren. Die Klasse *R* wird beim Erstellen eines neuen Android-Projektes in der Entwicklungsumgebung automatisch angelegt und ist für die Einbindung der Layout-Konfigurations-Dateien während der Laufzeit zuständig. Jedem Element muss dafür eine eindeutige Bezeichnung, die *id,* in der entsprechenden Konfigurationsdatei angegeben werden. Mit dieser Technik ist es zur Laufzeit möglich, mit Hilfe der entsprechenden Objekte und deren Initialisierung mit den grafischen Elementen zu arbeiten. Unteranderem können so Eingabewerte ausgelesen und verändert werden. Die Initialisierung wird bei dem Start der Anwendung in der Methode *onCreate()*.realisiert. Die Verwendung von Objekten wie Buttons erfordert die Überwachung der Benutzereingaben, d.h. es müssen mit Hilfe der Methoden *onClick()* oder *onLongClick()* entsprechende Funktionalitäten implementiert werden. Diese Methoden werden durch Listener-Interfaces, wie dem *OnClickListener*, bereitgestellt. Weiterhin bietet die Oberklasse *Activity* Methoden für die Verwaltung des Lebenszyklus einer Anwendung. In diesem Fall wird die Methode *onActivityResult()*-Methode genutzt, die aufgerufen wird, wenn man nach Nutzung einer anderen Anwendung wieder zurück in die eigentliche Anwendung zurückkehrt. Die Auswertung der Informationen, die durch den Aufruf der Barcode-Anwendung „Zxing“ zurückgeliefert werden, erfordert den Aufruf der Methode *onActivityResult()*, um die Daten verwenden zu können. Weiterhin finden die Methoden *onPause()* und *onResume()* Verwendung. Diese Methoden werden implementiert, um den Zustand der Anwendung bei kurzzeitigen Verlassen zu speichern und bei einem Neustart zurückzusichern. Neben den spezifischen Methoden der Android-Umgebung sind eigene Funktionalitäten zu implementieren. Dazu zählen in diesem Fall die Methoden *fillSpinner()*, *fillObjectDialog(), fillConnectionDialog()* und *controlSearch().* Für die Initialisierung der Auswahl-Felder („Spinner“) beim Starten 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 des Network-Managers an. Die JSON-Objekte mit den Such-Ergebnissen werden an die Methoden fillObjectDialog() oder fillConnectionDialog() übergeben, um die entsprechenden Ausgabe-Fenster zu generieren.

Die Methoden für die Kommunikation mit dem Web-Service sind in die Klasse Network-Manager im Paket „de.seideman.dams.manager“ ausgelagert. Zur Überprüfung 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-Schnittstellen gesendet wird. Weiterhin wird die Methode *readStream()* benötigt, um die Antworten des Web-Servers einzulesen.

Nachfolgendend soll an einem kurzen Quellcode-Ausschnitt die Realisierung der Kommunikation mit dem Web-Service aufgezeigt werden. Als Beispiel wird in **Fehler! Verweisquelle konnte nicht gefunden werden.** 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 bei Methoden-Aufruf durch die Methode *makeHash(pass)* konvertiert.

*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 : Methode tryLogin() aus der Klasse NetworkManager

Wichtige Bestandteile stellen die Objekte der Klassen HttpClient und HttpPost dar. Das neue Objekt der Klasse HttpPost wird sofort mit der URI

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

zur passenden 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. 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.

Abschließend soll mit Hilfe eines Sequenzdiagramms beispielhaft der Ablauf der Anwendung und die Kommunikationswege mit dem Web-Service skizziert werden.



Abbildung : Sequenzdiagramm "Login" und "Suche starten"

Als Grundlage für das Diagramm dienen die Anwendungsfälle Anwendung starten und 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“) und deren Unterklassen zu unterscheiden. Aufgrund der besseren Übersichtlichkeit wird bei den Methoden-Aufrufen auf die Eingabe-Parameter verzichtet.

Der Start der Anwendung wird durch den Akteur „Benutzer“ initiiert. Die Anwendung überprüft ob eine Netzwerk-Verbindung zum Web-Service besteht. Ist die Anwendung erfolgreich gestartet, kann der Benutzer seine Anmelde-Daten eingeben und mit dem Button „Einloggen“ einen Login-Versuch unternehmen. Die mobile Anwendung ruft dazu mit den Eingabe-Parametern die Methode *tryLogin()* der Klasse NetworkManager auf. Diese Methode generiert den HttpPost-Request und schickt die Anfrage an die Methode *tryLogin()* der Klasse AndroidService. Anschließend wird die Methode *login()* aufgerufen, die wiederum dafür Sorge trägt, dass die aktuelle Benutzerliste geladen wird. Die im HttpPost-Request gesendeten Anmelde-Daten werden intern verglichen und der Login Status über die einzelnen Instanzen zurück an die mobile Anwendung geschickt. Ein erfolgreicher Anmelde-Versuch führt zur Anpassung des Layouts, d.h. der Hauptbildschirm der Anwendung wird dargestellt. Wie in den Anwendungsfällen (siehe 3.1.3) bereits beschrieben hat der Benutzer nun die Möglichkeit den gewünschten Such-Parameter-Typ auszuwählen und das Such-Kriterium einzutragen. Mit dem Button „Suchen“ wird die Methode *getObjectInfo()* der Klasse NetworkManager aufgerufen und die Eingabe-Parameter übergeben. Auch hier wird ein HttpPost-Request erzeugt und die URI der entsprechenden Methode aufgerufen. Die Methode *getObjectInfo()* der Klasse AndroidService wertet die übergeben Parameter aus. Der vorliegende Fall zeigt eine Anfrage, in der der Benutzer eine Suche anhand der Seriennummer durchführen will. Aus diesem Grund wird die Methode *getObjectBySerial()* der Klasse ObjectManager aufgerufen und bei einer erfolgreichen Suche das Objekt vom Datentyp SapObject an die aufrufende Instanz zurückgegeben. 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 erzeugt und den Dialog darstellt.

I

## Testen der Android-Anwendung

### Zieldefinition

In diesem Kapitel wird die praktische Umsetzung von Unit-Tests in Bezug auf die Android-Umgebung näher betrachtet. Die Realisierung erfolgt in Anlehnung an das Tutorial „Activity Testing“ der offiziellen Developer-Webseite von Android [AND11] Dabei soll vor allem auf das android-spezifischen Verhalten der Anwendungen eingegangen werden. Dazu zählen unteranderem das Testen des Lebenszyklus einer Anwendung, die Initialisierung von GUI-Komponenten und das Testen der Anwendungsberechtigungen. Aufgrund der trivialen Struktur der mobilen Anwendung kann nur auf das Szenario „Activity Testing“ (siehe 2.3.4.2) eingegangen.

### Umsetzung

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

Abbildung : KLassendiagramm der Test-Klassen

Durch die Entwicklungsumgebung wurden die Klassen „DamsActivityTest“ und „LoginActivityTest“ als Testklassen zu den gleichnamigen Activity Klassen der eigentlichen 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 TestKlassen werden Instanzen der zu testenden Klassen generiert, die die Grundlage 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 (siehe 2.3.3) 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 und gegeben falls durch ein Listing beschrieben.

* testPreCondition()

Diese Test-Methode wird durchgeführt, um die erfolgreiche Inititlaisierung 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, was dazu führt, 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() kann abschließend geprüft werden, 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 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() zurück gesichert werden. In dieser Methode findet die Klasse *Instrumentation* (siehe 2.3.4.1) 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)* die Inhalte der grafischen Elemente mit den Konstanten initialisiert bzw. die vordefinierten Spinner-Positionen gesetzt. Nach dem die onPause() aufgerufen wurde, werden die Spinner-Positionen verändert und der Text des Eingabe-Feldes gelöscht. Anschließend wird die Methode *inst.callActivityOnResume(damsActivity) ausgeführt* und dann mit Hilfe der Methode assertEquals() geprüft, ob die nach dem Neustart der Anwendung zurückgesicherten Werte mit den Werten vor der Beendigung übereinstimmen.

* testOnDestroy()

Die Methode *testOnDestroy()* ist ähnlich implementiert wie die Methode *testOnPause()*. 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 die Klasse *ActivityInstrumentationTestCase2* zur Verfügung gestellt wird.

* testScanResult()

Unter Verwendung der Methode *testScanResult()*, kann die Funktionalität der Methode *onActivityResult()* überprüft werden. Diese Methode wird aufgerufen werden, nachdem der Benutzer mit Hilfe der Anwendung „Zxing“ einen Strichcode erfolgreich gescannt hat. Um das erfolgreiche Einlesen überprüfen zu können, wird ein *Intent* erzeugt, der die Informationen der Barcode-Anwendung speichert und als Paramater für die *onActivityResult()*-Methodebereitstellt. 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 Anwendung 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 gewährt werden.

* testItemSelected()

Für die Auswahl entsprechender Such-Kriterien 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* 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. (siehe 2.3.3) In diesem Test wird der für den Test der Netzwerk-Verbindung 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 erwaretet 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 Erstellung der mobilen Anwendung und es erfolgt ein Ausblick auf mögliche Erweiterungsmöglichkeiten.

# Abschluss

## Erweiterungsmöglichkeiten

Bei einer weiterführenden Implementierung sind verschiedene Erweiterungen der mobilen Anwendung möglich. Denkbar sind die in 3.1.3 erarbeiteten Anwendungsfälle des Akteurs „Dams-Admin“. Für den Anwendungsfall „Inventur“ ist vorstellbar, die Daten der Web-Anwendung mit den IST-Daten in den Rechenzentren zu vergleichen und über die Auswertung ein Änderungsprotokoll zu erstellen. Weiterhin ist denkbar mit Hilfe der mobilen Anwendung ein neues Gerät in den Rechenzentren aufzunehmen und zu platzieren. Zur besseren Nutzbarkeit ist auch die Realisierung einer Visualisierungsfunktion möglich, um das Auffinden gesuchter Komponenten zu vereinfachen. Für eine bessere Performance beim Einscannen von Strichcodes ist es denkbar, von der mobilen Anwendung abgesetzte Bluetooth-Handscanner einzusetzen.

## Fazit