Inhaltsverzeichnis

1. Einleitung 2

1.1. Abstract 2

1.2. Vorgehensweise und Methodik 2

1.3. Projektumgebung 2

1.4. Motivation 2

1.5. Aufgabenstellung 2

1.6. Abgrenzung der Projektaufgabe 2

2. Theoretische Vorbetrachtung 2

2.1. Allgemein 2

2.2. Zielplattform Android 2

2.3. Webservice 2

2.4. Software Tests 2

2.4.1. Einführung 2

2.4.2. Unit-Test 2

2.4.3. JUnit 2

2.4.4. Testen in Android 2

2.4.4.1. Grundlagen 2

2.4.4.2. Activity Testing 2

2.4.4.3. Service Testing 2

2.4.4.4. Content Provider Testing 2

3. Umsetzung 2

3.1. Analyse „Data Asset Management Solution“ 2

3.1.1. Webanwendung 2

3.1.2. Datenbank 2

3.1.3. Anwendungsfälle konzipieren 2

3.2. Realisierung des Webservice 2

3.2.1. Zieldefinition 2

3.2.2. Planung 2

3.2.2.1. Datenbankzugriff 2

3.2.2.2. Aufbereitung der Daten 2

3.2.2.3. Bereitstellung der Schnittstellen 2

3.2.3. Umsetzung 2

3.3. Realisierung der Anwendung 2

3.3.1. Zieldefinition 2

3.3.2. Planung 2

3.3.2.1. Layout 2

3.3.2.2. Activities 2

3.3.2.3. Zugriff auf den Webservice 2

3.3.3. Umsetzung 2

3.4. Testen der Android-Anwendung 2

3.4.1. Zieldefinition 2

3.4.2. Planung 2

3.4.3. Umsetzung 2

4. Abschluss 2

4.1. Erweiterungsmöglichkeiten 2

4.2. Ausblick 2

# Einleitung

## Abstract

## Vorgehensweise und Methodik

## Projektumgebung

## Motivation

Die Verwaltung eines hochproduktiven Rechenzentrums ist sehr komplex, da dafür unterschiedlichsten Daten der verbauten Komponenten erfasst werden müssen. Naturgemäß ist dabei auch die Fehlersuche während der laufenden Produktion hochzeitkritisch. Um diese Aufgaben bewältigen zu können wurde die in PHP geschriebene Webanwendung „speedikon DAMS“ (Data Center Asset Management Solutions) beschafft. Diese Softwarelösung unterstützt die Verwaltung und Visualisierung von Bestandsdaten in den Rechenzentren. Die Pflege der Daten erfolgt im Moment ausschließlich über eine Weboberfläche, die lediglich einen PC-Arbeitsplatz mit entsprechendem Webbrowser voraussetzt. Um die Nutzbarkeit der bereits aufgenommen Daten flexibler zu gestalten, soll nun zusätzlich eine portable Lösung geschaffen werden, die mit kurzen Zugriffszeiten die technischen Daten, Verkabelungswege und Standorte der einzelnen Komponenten hausweitwiedergeben kann.

## Aufgabenstellung

In diesem Projekt soll ein Prototyp einer Android-Anwendung entwickelt und implementiert werden, der die vorhandenen Daten von „speedikon DAMS“ nutzt. Die vorhandene Webanwendung stellt keine Schnittstellen für den Abruf der Daten zur Verfügung. Aus diesem Grund soll ein Webservice entworfen und implementiert werden, der diese Aufgabe übernimmt. Die mobile Anwendung soll eine Benutzeroberfläche zur Verfügung stellen, die nach einer erfolgreichen Anmeldung des Nutzers die Suche und Abfrage von Geräteinformationen mit Hilfe diverser Suchkriterien ermöglicht und eine entsprechende Ausgabe generiert.

## Abgrenzung der Projektaufgabe

Aufgrund der fehlenden Schnittstellen für die Manipulation der Datensätze und der Komplexität der Webanwendung, soll die mobile Anwendung ausschließlich lesenden Zugriff auf die Datensätze erhalten. 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-Netz des Verlages.

# Theoretische Vorbetrachtung

## Allgemein

## Zielplattform Android

## Webservice

Wie in der Aufgabenstellung beschrieben, stellt die zugrunde liegende Web-Anwendung keine Schnittstellen für den Zugriff auf die vorhandenen Daten zur Verfügung, so dass ein entsprechender Webservice 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 Prototocol (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 Annotations zu verwenden, um Metadaten die für den Webservice benötigt werden einbinden zu können. Im Folgenden sollen kurz die wichtigsten Annotations aufgezeigt werden.

@Path

Definiert den Pfad mit der die zur Verfügung gestellte Klasse aufgerufen werden kann und bildet so den Knotenpunkt („root ressource“) 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 Annotationen 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 Klammer 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:

* Unittest

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

* Integrationstest

Diese Tests stellen die nächste Stufe der Softwaretests nach den Unittests dar. Im Gegensatz zu den Unittests, 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 CPU5. 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 soll die Betrachtung 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 sollte automatisch erfolgen. Der Hintergrund dafür besteht in der Wiederverwendbarkeit der Tests, die sofort aufzeigen sollen falls ein Tests 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 (siehe 2.4.4.1)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 2.4.3)genutzt werden, um die Testumgebung nach jeder Testklasse zurückzusetzen.

* Professionel

### JUnit

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 Unittests 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 Exceptions angesehen werden. Das folgende Code-Beispiel soll die Funktionweise 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 beschrieben Assert-Methoden des JUnit-Framework 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-Objekt 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 gegensietig 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 Abhängikeiten einer Anwendung zu anderen Ressourcen abzubilden. So besteht die Möglickeit, 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 vorallem in der Laufumgebung, 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-Anwednung 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-Frame work. 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“

### Webanwendung

### Datenbank

### Anwendungsfälle konzipieren

## Realisierung des Webservice

### Zieldefinition

### Planung

#### Datenbankzugriff

#### Aufbereitung der Daten

#### Bereitstellung der Schnittstellen

### Umsetzung

## Realisierung der Anwendung

### Zieldefinition

### Planung

#### Layout

#### Activities

#### Zugriff auf den Webservice

### Umsetzung

## Testen der Android-Anwendung

### Zieldefinition

### Planung

### Umsetzung

# Abschluss

## Erweiterungsmöglichkeiten

## Ausblick