Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Andre Ziehn

Entwurf und Realisierung eines Clicktrackers zur Unterstützung interaktiver Vorlesungen unter Android

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andre Ziehn

Entwurf und Realisierung eines Clicktrackers zur Unterstützung interaktiver Vorlesungen unter Android

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

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

Eingereicht am: 23. August 2012

Andre Ziehn

Thema der Arbeit
Entwurf und Realisierung eines Clicktrackers zur Unterstützung interaktiver Vorlesungen unter Android

Stichworte
Classroom Response System, Android, interaktive Vorlesung, verteiltes System

Kurzzusammenfassung
Classroom Response Systems stellen Systeme dar, die es Vortragenden einer Vorlesung ermöglichen, die Zuhörer durch technische Hilfsmittel aktiv einzubeziehen. Grundgedanke dieser Systeme ist die Verbesserung von Vorlesungen durch die Ergänzung von interaktiven Inhalten.

In dieser Arbeit wird ein Classroom Response System mit dem Ziel entwickelt, Zuhörern die aktive Teilnahme mit mobilen Endgeräten, wie z.B. Smartphones, Tablets oder Laptops, zu ermöglichen. Beispielhaft wird dies für Geräte mit dem Android-Betriebssystem realisiert.

Andre Ziehn

Title of the paper
Design and implementation of a click tracker to support interactive lectures on Android

Keywords
classroom response system, android, interactive lecture, distributed system

Abstract
Classroom response systems depict systems, which allow a lecturer to actively involve audience with technical tools. The basic principle of such systems is the improvement of lectures by supplementing interactive content.

This thesis deals with the development of a classroom response system with the objective of enabling active involvement of audience with mobile devices, such as smart phones, tablets or notebooks. This is exemplified by the implementation for Android devices.

Inhaltsverzeichnis

1 Einleitung

In diesem Kapitel folgt eine kurze Einführung, die die Motivation, das Ziel und den Aufbau der Arbeit erläutert.

1.1 Motivation

Interaktive Vorlesungen haben sowohl für den Professor als auch für Studenten Vorteile gegenüber normalen Vorlesungen. Professoren wird die Möglichkeit gegeben, das Verständnis wichtiger Lehrinhalte zu überprüfen und sie so weiter zu festigen oder gegebenenfalls zu wiederholen. Studenten haben den Vorteil, nicht durchweg passiv zuhören zu müssen, und können so ihre Konzentration einfacher aufrecht erhalten. Auch wird zurückhaltenden oder unsicheren Studenten die Möglichkeit geboten, anonym ihre Lösung und Meinung mitzuteilen.

Weiterhin besteht für den Professor die Möglichkeit, direkt Feedback für seine Vorlesung zu bekommen und Probleme so frühzeitig zu erkennen und zu lösen.

1.2 Zielsetzung

Die Arbeit hat das Ziel, Vorlesungen durch den Einsatz von mobilen Endgeräten interaktiver zu gestalten. Dafür wird ein sogenanntes „Clicktracker“-System (auch „Classroom Response System“ genannt) entworfen und realisiert.

Es wird eine Anwendung für den Vortragenden einer Vorlesung entwickelt, die diesem das Erstellen von interaktiven Inhalten, wie beispielsweise Multiple-Choice-Fragen oder zu lösenden Rechnungen, ermöglicht und die Ergebnisse visualisiert.

Für die Teilnehmer wird eine Anwendung entwickelt, die diesen ermöglichen soll, an den interaktiven Inhalten anonym teilzunehmen. Realisiert wird diese Anwendung unter Android. Dies stellt dabei aber nur einen beispielhaften Client dar. Das System soll so entworfen werden, dass Clients für andere Plattformen ebenfalls integriert werden können.

Im Hintergrund wird ein Server arbeiten, der das Bindeglied der beiden eben beschrieben Anwendungen darstellt. Über diesen wird kommuniziert und er speichert die erstellten interaktiven Inhalte sowie deren Ergebnisse.

1.3 Gliederung

Die gesamte Arbeit teilt sich in acht Kapitel auf. Nach diesem einleitenden Kapitel folgt ein Grundlagen-Kapitel (Kapitel 2). Hier geht es um eine Einführung in das Thema und einige, in dieser Arbeit verwendeten, grundlegende Techniken. Es beginnt mit einer Einführung zu „Classroom Response Systems“ und interaktiven Vorlesungen. Nach dieser grundlegenden Einführung zum Hintergrund dieser Arbeit folgt eine technische Einführung zu Android, Webservices und der Java EE Plattform.

Kapitel 3 dient der Analyse. Es werden sowohl vorhandene Lösungen betrachtet als auch Interviews mit potenziellen Nutzern geführt. Darauf aufbauend werden Anforderungen ermittelt, welche die Grundlage für die Spezifikation schaffen.

In Kapitel 4 folgt die Spezifikation. Es wird ein fachliches Datenmodel entworfen sowie grundlegende Anwendungsfälle und Dialoge beschrieben. Hierauf aufbauend wird das System in Kapitel 5 entworfen. Dieses Kapitel teilt sich in drei Sektionen. Es werden Komponenten gesucht und beschrieben, der Webservice und dessen Kommunikationsprotokoll entworfen und schließlich die Architektur des gesamten verteilten Systems entworfen.

In Kapitel 6 wird die Realisierung beschrieben. Es wird hierbei zwischen den verschiedenen Teilen der Anwendung, der Server-, Studenten- und Professor-Anwendung unterschieden, da diese auf verschiedenen Knoten und unter anderen Systemen laufen. Die Tests der Realisierung werden in Kapitel 7 beschrieben.

Abschließend folgt in Kapitel 8 eine Zusammenfassung und ein Ausblick über Erweiterungs- und Verbesserungsmöglichkeiten des Systems.

2 Grundlagen

In diesem Kapitel werden grundlegende Themen beschrieben, die für das Verständnis dieser Arbeit wichtig sind. Begonnen wird mit einer Erklärung von „Classroom Response Systems“ und einer Beschreibung der generellen Motivation von interaktiven Vorlesungen.

Weiterhin werden Android, Webservices und Java EE näher betrachtet, da diese die Grundlagen des Systems dieser Arbeit darstellen.

2.1 Classroom Response Systems

In diesem Abschnitt folgt eine Einführung zu Classroom Response Systems und interaktiven Vorlesungen.

2.1.1 Einführung

Der Begriff „Classroom Response Systems“ (CRS) steht für Systeme, die es Vortragenden einer Vorlesung ermöglichen, die Zuhörer durch technische Hilfsmittel aktiv einzubeziehen (s. Bruff, 2010).

Eine aktive Teilnahme von Zuhörern kann in verschiedenen Formen erfolgen. Die einfachste Form wäre eine Abstimmung per Handzeichen. Dies erfordert wenig Aufwand in der Vorbereitung, dafür umso mehr bei der Auswertung der Handzeichen. Der nächste Schritt sind vom Zuhörer zu lösende Aufgaben. Dies bereitet ebenfalls wenig Aufwand in der Vorbereitung und ermöglicht darüber hinaus erweiterte Fragemöglichkeiten gegenüber der Abstimmung per Handzeichen. Nachteil ist hier die Auswertung, da nur der kleinste Teil der Lösungen der Zuhörer einbezogen werden kann, sofern die Lösungen nicht eingesammelt werden. Dies würde aber erhöhten Aufwand bei der Auswertung bedeuten und keine sofortige Auswertung ermöglichen. Ebenfalls bieten beide Lösungen keinerlei grafische Visualisierung der Ergebnisse.

Um die Vorteile dieser Möglichkeien der aktiven Teilnahme zu vereinen, wurden Systeme entwickelt, die „Classroom Response Systems“ genannt werden. Bei dieser Form der aktiven Teilnahme gibt es eine Möglichkeit für den Professor, Fragen digital vorzubereiten und während

einer Vorlesung abzurufen. Studenten brauchen entsprechende Endgeräte, um diese Aufgaben während der Vorlesung zu beantworten. Diese Systeme erfassen sämtliche Antworten, können diese auswerten und die Ergebnisse visualisieren. Es gibt verschiedene Ansätze um diese Idee umzusetzen. Sie unterscheiden sich in der Art des Endgerätes, welches die Zuhörer nutzen. Es gibt einfach gehaltene Hardware-Geräte, die eine Anzahl von Knöpfen für die entsprechenden Antworten bereitstellen (Beispielsweise A bis D, um vier Antwortoptionen zu ermöglichen). Weiter gibt es Geräte, die ein Display besitzen, was die Möglichkeiten der Fragetypen und die Anzahl der möglichen Antwortoptionen erhöht. Schlussendlich gibt es aber auch Systeme, die auf Hardware setzen, die die meisten Studenten besitzen. Dies wären Laptops, Smartphones oder Tablets. Hier gibt es keine Einschränkungen hinsichtlich der Art der möglichen Antworttypen.

Ein weiterer Punkt, der die Ansätze unterscheidet, ist die Form der Kommunikation der Endgeräte der Studenten mit dem Endgerät des Professors. Entsprechend der eben vorgestellten Möglichkeiten wären die Verbindung per Infrarot, per Internet oder per SMS möglich. Die Wahl des Mediums hängt von den technischen Gegebenheiten des jeweiligen Einsatzortes ab.

Die Nutzung von „Classroom Response Systems“ ist ein interessanter Ansatz um Teilnehmer einer Vorlesung einzubeziehen. Warum dies Sinn macht wird im nächsten Abschnitt näher erläutert.

2.1.2 Interaktive Vorlesungen

Interaktive Inhalte ermöglichen aktives Lernen, was Studenten in Vorlesungen unterstützen kann. Arthur W. Chickering beschreibt sehr treffend, was interaktives Lernen bedeutet und warum dies sinnvoll ist. „Learning is not a spectator sport. Students do not learn much just by sitting in classes listening to teachers, memorizing pre-packaged assignments, and spitting out answers. They must talk about what they are learning, write about it, relate it to past experiences, apply it to their daily lives. They must make what they learn part of themselves.“ (Chickering und Gamson, 1987)

Weiterhin schreibt Chickering Feedback eine große Wirkung zu. Studenten sollen ihr Wissen möglichst oft reflektieren können, um besser einschätzen zu können, was sie wie gut verinnerlicht haben. „Knowing what you know and don’t know focuses learning“ (Chickering und Gamson, 1987) beschreibt dieses Ziel sehr gut.

Unter der Prämisse des Willens zum Lernen kann man das staatliche Schulsystem in Deutschland betrachten und findet genau diese Punkte wieder. Aktives Lernen wird durch mündliche Mitarbeit, Schülervorträge oder auch einfach das Präsentieren einer Lösung an der Tafel erreicht. Da man bedingt durch die mündliche Benotung sowohl ungefragt als auch

gewollt Fragen beantwortet, reflektiert man sein Wissen. Dazu tragen auch mehrere Klausuren und Tests pro Schuljahr bei. An Universitäten und Fachhochschulen kann dies bedingt durch die hohe Teilnehmerzahl einer Vorlesung von Professoren nicht bewältigt werden. Hier müssen alternative Wege gefunden werden, um aktives Lernen zu ermöglichen und einer davon sind die im vorherigen Abschnitt beschriebenen „Classroom Response Systems“.

2.2 Android

Android ist die Grundlage der Studenten-Anwendung, die in dieser Arbeit entwickelt wird und wird daher im Folgenden beschrieben.

2.2.1 Einleitung

Android ist ein Betriebssystem für Geräte mit Mikroprozessoren. Ursprünglich für Smartphones gedacht, läuft es inzwischen auf den meisten Geräten, die auf ARM-, MIPS- oder X86-Prozessoren aufbauen (z.B. Netbooks, Tablets, Set-Top-Boxen oder auch Auto-Infotainment-Systeme) (Becker und Pant, 2010, S. 19).

Der Grundgedanke von Android ist die offene Entwicklung, welche es Entwicklern ermöglicht, alle Optionen der verwendeten Geräte zu nutzen. Es wird quelloffen entwickelt und basiert auf einem Linux-Kernel.

2.2.2 Architektur

Die Android-Systemarchitektur ist in Abbildung 2.1 übersichtlich dargestellt und wird im folgenden näher erläutert.

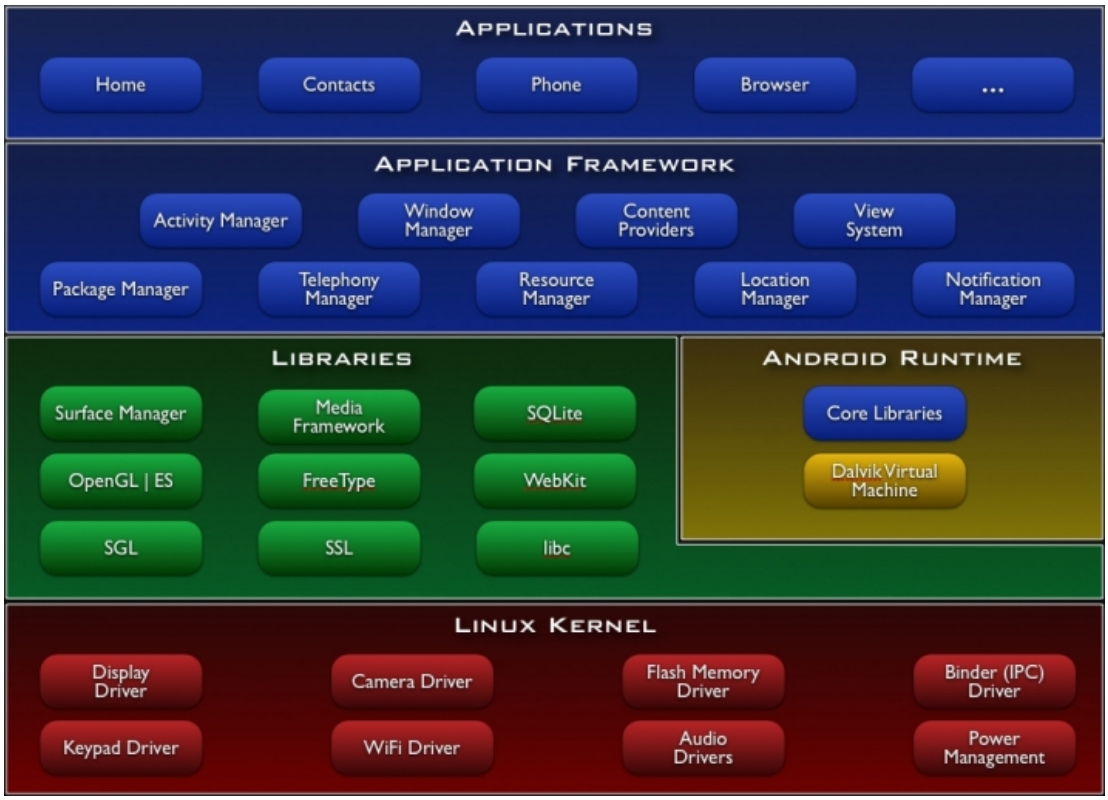

Abbildung 2.1: Die Android-Systemarchitektur (Google Inc., 2012b)

Linux

Android basiert auf der Version 2.6 des Linux-Kernels. Er stellt die Abstraktionsschicht zwischen der Hard- und der Software dar, da er die Gerätetreiber enthält.

Der Kernel ist für den Einsatz auf Mikroprozessoren hinsichtlich Energieverbrauch und Speichermanagement optimiert.

Laufzeitumgebung

Die Laufzeitumgebung von Android basiert auf der DVM (Dalvik Virtual Machine), welche im Hinblick auf die Anforderungen von mobilen Geräten entwickelt wurde. Sie basiert auf der JVM (Java Virtual Machine) Apache Harmony, einer Open-Source-Version von Java, enthält aber einige signifikante Änderungen (Becker und Pant, 2010, S. 23).

Google verwendet für die DVM einen eigenen Dalvik-Bytecode. Abbildung 2.2 zeigt den Weg vom Java-Quellcode zum Dalvik-Bytecode. Der Quellcode wird mit einem Java Compiler in Java-Bytecode gewandelt. Diesem Bytecode kann der für die DVM entwickelte Compiler

„dx“ in Dalvik-Bytecode umwandeln. Der zweite große Unterschied zu Apache Harmony ist die Optimierung für Mikroprozessoren. Diese besitzen Register, welche Berechnungen enorm beschleunigen können, und die DVM nutzt diese aus.

Ein Vorteil, den Apache Harmony bereits enthielt, ist die Optimierung hinsichtlich eines kleinen Speicherverbrauchs und der Möglichkeit, viele Instanzen der JVM parallel laufen zu lassen. Dies nutzt Android, da für jede Anwendung ein eigener Prozess im Betriebssystem gestartet wird, in welchem wiederum eine DVM gestartet wird, in der dann die Anwendung läuft. Größte Vorteile bei diesem Vorgehen sind die Sicherheit und Verfügbarkeit.

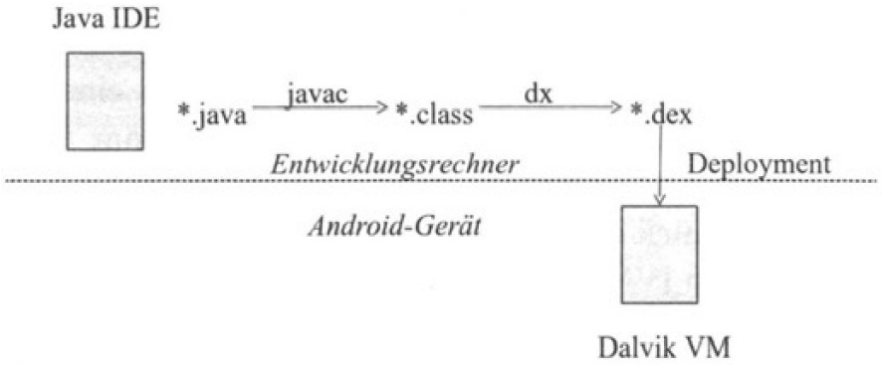

Abbildung 2.2: Von *.java zu *.dex (Becker und Pant, 2010)

Bibliotheken

Android enthält eine Menge von Standardbibliotheken, welche in C/C++ geschrieben sind. Diese bieten alle Funktionen, die für Android-Anwendungen erforderlich sind (z.B. Datenbanken oder 2-D- bzw. 3-D-Grafikbibliotheken). (Google Inc., 2012b).

Anwendungsrahmen

Der Anwendungsrahmen ist der Unterbau für Android-Anwendungen. Er abstrahiert die Hardware und bietet sie in Form von Manager-Klassen an, welche komplett in Java geschrieben sind.

Dieses Design ist auf die einfache Wiederverwendung von Komponenten ausgelegt. Jede Anwendung kann ihre Funktionen zur Verfügung stellen, damit diese von anderen verwendet werden können.

Unter Android gibt es vier zentrale Komponenten, aus denen sich Anwendungen zusammensetzen können (Becker und Pant, 2010, S. 25). Diese werden im nächsten Abschnitt 2.2.3 näher beschrieben.

Anwendungsschicht

In der Anwendungsschicht befinden sich die in Java geschriebenen Anwendungen. Hier sind sowohl Standardanwendungen von Google (z.B. Email-Client, Browser oder Kontakte) als auch eigene Anwendungen zu finden.

Sowohl die Interaktion von Menschen mit der Anwendung als auch die Kommunikation der Anwendungen untereinander finden auf dieser Ebene statt.

2.2.3 Android-Komponenten

Wie bereits beim Anwendungsrahmen (siehe 2.2.2) beschrieben, wird bei Android komponentenbasiert gedacht und entwickelt. Die vier zentralen Komponenten, aus denen sich Anwendungen zusammensetzen können, werden im Folgenden kurz vorgestellt.

Activity Activities sind für die Oberflächen einer Anwendung zuständig. Sie stellen diese dar, verwalten sie und reagieren auf Interaktionen vom Anwender.

Service Die Service-Komponente ist für das Verwalten von Hintergrundprozessen zuständig. Sie ist unabhängig von Activities, da nicht immer Oberflächen benötigt werden.

Content Provider Content Provider sind für die Verwaltung von Daten zuständig. Sie werden lose an Anwendungen gekoppelt und abstrahieren die darunterliegende Persistenzschicht.

Broadcast Receiver Broadcast Receiver sind in der Lage Systemnachrichten zu empfangen, über welche Änderungen des Systemzustandes kommuniziert werden können.

2.3 Webservice

Das World Wide Web Consortium (W3C) hat eine Definition eines Webservices veröffentlicht:

> „A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically

> conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“ (Booth u. a., 2004)

Weiterhin beschreibt das W3C, dass Anwendungen, welche über das Internet operieren und auf verschiedenen Plattformen arbeiten, für Webservices besonders geeignet sind. Diese zwei Kernpunkte der verteilten Anwendung, die Thema dieser Arbeit ist, sprechen für den Einsatz eines Webservices. Nicht alle Ansätze für die Umsetzung eines Webservice folgen komplett der Definition des W3C, sondern erweitern und verbessern viel mehr die grundlegenden Ansätze.

Im Folgenden wird SOAP[1] erläutert. Dies ist einer der bekanntesten und verbreitetsten Ansätze, welcher der Definition des W3C folgt.

SOAP

SOAP ist die Definition eines Protokolls zum Austauschen von Daten, basierend auf XML als Nachrichtenformat. Die Struktur der Daten wird mit Hilfe der WSDL (Web Service Description Language) festgelegt. Dies ist nicht zwingend erforderlich, sollte aber nur weggelassen werden, wenn die Struktur hinreichend bekannt ist (vgl. Gudgin u. a., 2007).

Die Übertragung der Nachrichten ist unabhängig vom Transportprotokoll. Es können beispielsweise HTTP oder SMTP verwendet werden.

Die größten Vorteile von SOAP sind die Plattformunabhängigkeit und die durch das W3C durchgesetzte Standardisierung. Nachteile ergeben sich durch XML, da Nachrichten in diesem Format schnell sehr groß werden und der dadurch entstehende Overhead die Performanz negativ beeinflusst.

Ein weiterer bekannter Ansatz zur Umsetzung eines Webservices ist REST (Representational State Transfer). Er unterscheidet sich von der W3C-Definition im Nachrichtenformat, der Serialisierung und der optionalen Beschreibung der Schnittstellen.

REST-Architekturstil

REST ist ein Architekturstil, welcher von Roy T. Fielding im Rahmen seiner Dissertation (Fielding, 2000) aus der konkreten Architektur von HTTP abstrahiert wurde. Das World Wide Web (WWW) ist folglich eine Ausprägung dieses Architekturstils.
Dieser Ansatz basiert auf fünf Prinzipien (s. Tilkov, 2011, S. 11):

[1] Wurde ursprünglich als Akronym für „Simple Object Access Protocol“ verwendet. Dies gilt seit Version 1.2 nicht mehr, da es nicht nur Zugriff auf Objekte ermöglicht.

1 Ressourcen mit eindeutiger Identifikation

Basierend auf dem Konzept des WWW werden Ressourcen mit Hilfe von URIs eindeutig identifiziert. Dies hat den Vorteil, dass ein bereits bestehendes und konsistentes globales Schema für Namen verwendet wird, welches sehr skalierbar ist.

Beispiel-URI für eine Ressource „Vorlesung“:
http://www.name.com/clicktracker/vorlesung/VL-1

2 Verknüpfungen

Verknüpfungen sind ebenfalls bereits aus dem WWW bekannt. Der große Vorteil von Verknüpfungen ist das unter Punkt 1 angesprochene globale Namensschema, durch welches Ressourcen jeglicher Art und unabhängig vom Standort miteinander verknüpft werden können. Weiterhin kann über Verknüpfungen der Anwendungszustand gesteuert werden.

Als Beispiel dient wieder die Vorlesungs-Ressource, welche eine Verknüpfung enthalten könnte, über die sie gelöscht werden kann. Ist dies nicht möglich, wäre dieser Link entsprechend nicht enthalten. Verknüpfungen sind folglich ein guter Ansatz um verfügbare Statusübergänge zu zeigen.

3 Standardmethoden

Kernidee von REST ist eine immer geltende Schnittstelle für Ressourcen. Es gibt also eine Grundmenge von Operationen, die für alle Ressourcen gültig sind. Um welche Methoden es sich hierbei handelt ist nicht festgelegt. Bei der Umsetzung der Architektur mittels HTTP werden diese Standardmethoden durch Standardverben von HTTP definiert. Diese sind GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE und CONNECT.

Daraus folgen die Vorteile, die diese Operationen mit sich bringen. Als Beispiel sei hier die GET-Methode genannt, die ein effizientes Caching unterstützt und idempotent (s. Idempotenz) ist.

4 Unterschiedliche Repräsentationen

Es besteht die Möglichkeit, für eine Ressource mehrere unterschiedliche Repräsentationen zur Verfügung zu stellen. Dies macht entsprechend verschiedener Anforderungen Sinn.

Als Beispiel sei eine Ressource gegeben, für die es zu einer XML- oder JSON (JavaScript Object Notation)-Repräsentation auch eine entsprechende HTML-Repräsentation gibt. Dies wäre sinnvoll, um die Ressource auch aus einem Webbrowser erreichbar zu machen.

5 Statuslose Kommunikation

Aus Gründen der Skalierbarkeit und des Ziels der möglichst losen Kopplung zwischen Client und Server ist die Kommunikation statuslos. Der Server hält keinen Sitzungsstatus. Zustände müssen in eine Ressource umgewandelt werden oder im Client gehalten werden.

2.4 Java EE (Java Platform, Enterprise Edition)

Java EE ist eine Spezifikation für eine Standardarchitektur zur Entwicklung von verteilten, mehrschichtigen, skalierbaren, sicheren und verlässlichen Java Anwendungen. Vor dem Hintergrund immer geringeren Budgets, weniger Ressourcen und der Forderung nach schnellerer Fertigstellung bei der Anwendungsentwicklung in Unternehmen, versucht Java EE eine Unterstützung dieser Ziele durch die Bereitstellung von APIs zu schaffen. Der Fokus liegt hierbei, wie der Name sagt, auf Unternehmensanwendungen. Da Java EE prinzipiell eine Erweiterung von Java SE (Java Platform, Standard Edition) darstellt, können alle Java SE APIs auch unter Java EE verwendet werden.

Die Java EE Plattform verwendet für Anwendungen eine mehrschichtige Architektur. Wie in Abbildung 2.3 gezeigt, besteht diese Architektur aus drei, bzw. vier Schichten, verteilt auf drei verschiedene Einheiten:

Client Die Client-Komponente kapselt Client-Anwendungen, wie zum Beispiel Webanwendungen, die durch einen Browser gesteuert werden, Java-Applets oder dem Thema dieser Arbeit entsprechend, eine Anwendung für ein mobiles Endgerät.

Server Ein Java EE Server kapselt die Web- und die Geschäftsschicht der Anwendung. Clients wird die Möglichkeit geboten, mit der Webschicht oder direkt mit der Geschäftsschicht zu kommunizieren. Die Webschicht ist optional und kann Servlets oder JSP (JavaServer Pages) Seiten enthalten, die HTTP-Anfragen verarbeiten.

Die Geschäftslogik der Anwendung findet sich in der Geschäftsschicht wieder, welche wiederum mit der Schicht der Datenbank kommuniziert.

Datenbank Der Datenbank-Server kapselt die Verarbeitung von Transaktionen und das Datenbanksystem.

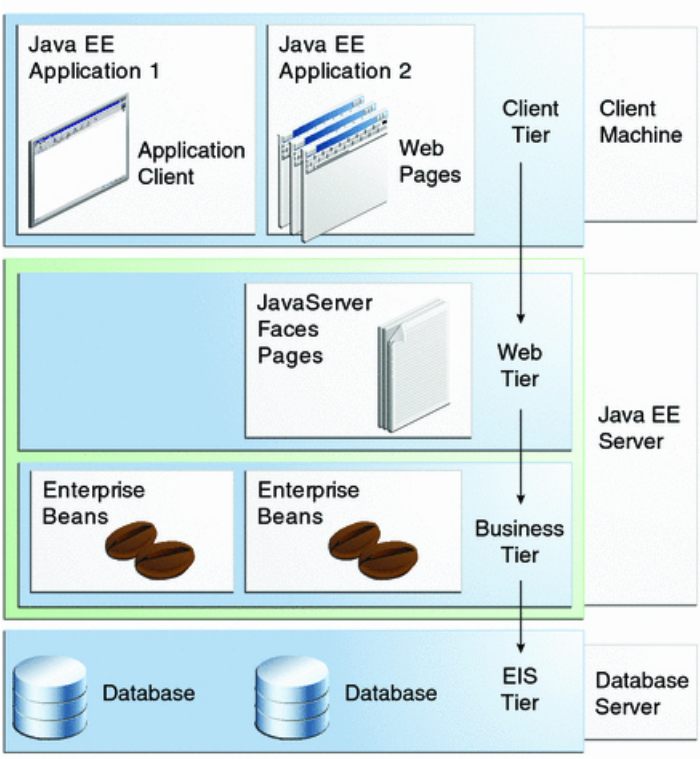

Abbildung 2.3: Mehrschichtige Java EE Architektur (Jendrock u. a., 2012, S. 43)

Desweiteren findet in der Infrastruktur von Java EE das Container-Entwurfsmuster Verwendung. Es gibt Komponenten und Container. Eine Komponente stellt eine in sich geschlossene voll funktionale Softwareeinheit dar, die mit anderen Komponenten interagieren kann (Jendrock u. a., 2012).

Container sind die Schnittstellen von Komponenten zu individuellen plattformabhängigen Funktionen, die sie mittels Diensten bereitstellen. Container abstrahieren damit technische Komplexität und sollen so den Fokus der Entwickler stärker auf die Geschäftslogik lenken (Goncalves, 2010, S. 6).

Eine Java EE Anwendung besteht aus vier Standard-Containern, die in Abbildung 2.4 dargestellt sind und im Folgenden kurz erläutert werden.

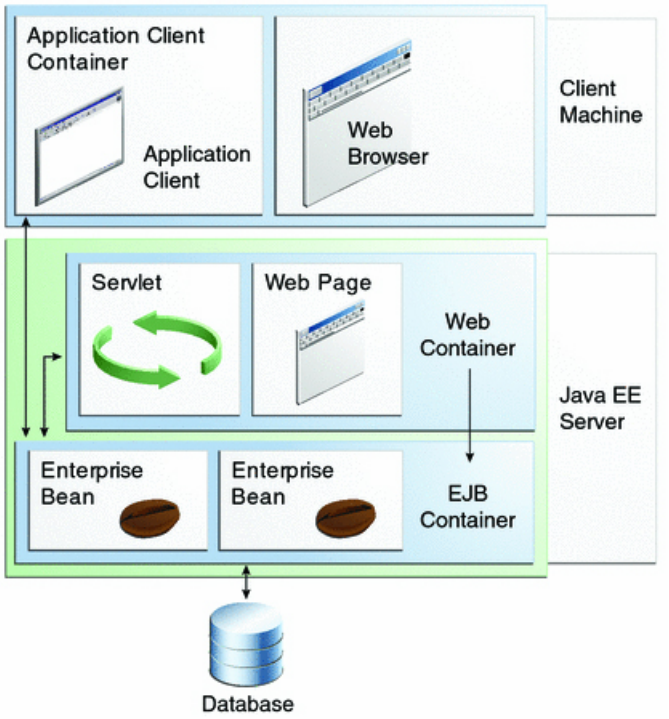

Abbildung 2.4: Java EE Container (Jendrock u. a., 2012, S. 50)

Applet-Container Der Applet-Container befindet sich in der Client-Schicht. Er ist für die Ausführung von Java-Applets zuständig, welche im Webbrowser laufen.

Application-Client-Container Der Application-Client-Container befindet sich ebenfalls in der Client-Schicht. Er verwaltet die Ausführung von Client-Programmen, welche direkt mit der Geschäftsschicht kommunizieren.

Web-Container Der Web-Container befindet sich in der Webschicht. Er ist für die Ausführung von Servlets und JavaServer Pages verantwortlich.

EJB-Container Der EJB-Container kapselt die Geschäftslogik einer Java EE Anwendung.

Es folgen einige Beispiele von angebotenen Diensten, die teilweise Verwendung in dieser Arbeit finden:

Java Persistence API (JPA) JPA stellt die Standard-API für die Persistenz dar. Sie basiert auf objektrelationalem Mapping. Die bekannteste vollständige Implementierung dieser API

stellt Hibernate[2] dar. Durch die Verwendung von Hibernate Annotations, Hibernate EntityManager und Hibernate Core entsteht eine vollständige JPA-Implementierung (Kehle u. a., 2008, S. 38).

Dependency Injection Java EE 6 bietet zwei Spezifikationen für Dependency Injection an. JSR 330 (Dependency Injection for Java), welche einen Standard an Annotationen für das Injizieren in Komponenten bietet und JSR 299 (Contexts and Dependency Injection), welches eine Menge von kontextabhängigen Diensten anbietet, um die Dependency Injection zu vereinfachen.

Eine Referenzimplementierung stellt das Weld-Framework[3] der Seam-Plattform dar.

RESTful Web Services Unter der Spezifikation JSR 311 (JAX-RS) ist eine API definiert, die die Entwicklung eines Webservices nach dem REST-Architekturstil unterstützt.

Eine Referenzimplementierung stellt das Jersey-Framework[4] dar.

[2] http://www.hibernate.org/
[3] http://seamframework.org/Weld
[4] http://jersey.java.net/

3 Analyse

In diesem Kapitel werden vorhandene Clicktracker-Lösungen analysiert und Interviews mit potenziellen Nutzern geführt, um daraus Anforderungen zu entwickeln.

3.1 Vorhandene Lösungen

Es gibt bereits einige CRS-Lösungen auf dem Markt. Als Beispiel dient das CRS der Firma H-ITT (s. H-ITT, 2010), welches die Universität Hamburg verwendet. Es baut auf mobilen Hardware-Endgeräten mit einer festen Anzahl an Knöpfen für Antworten auf. Diese kommunizieren über Radio-Frequenzen mit einer Empfänger-Einheit, welche wiederum mit dem Computer des Professors verbunden ist. Sowohl Fragen für Vorlesungen als auch deren Auswertungen werden lokal gespeichert und verwaltet.
Vorteile eines solchen Ansatzes:

- Unabhängigkeit von örtlichen Gegebenheiten
- Simple Bedienung für Studenten

Nachteile eines solchen Ansatzes:

- Hohe Anschaffungskosten
- Kaum Möglichkeiten zur Erweiterung des Systems
- Dezentrale Speicherung von Aufgaben und Ergebnissen erfordert erhöhten Aufwand und weniger Flexibilität für Professoren

Der Ansatz, der in dieser Arbeit verfolgt wird, soll genau diese Nachteile verbessern. Um die hohen Anschaffungskosten zu vermeiden und das System einfach erweiterbar zu machen, sollen Notebooks, Netbooks, Tablets oder auch Smartphones als Endgeräte verwendet werden können, da diese unter Studenten bereits sehr verbreitet sind (In dieser Arbeit wird eine Beispielimplementation eines Clienten für Android-Smartphones vorgenommen). Weiterhin wird über einen Webservice kommuniziert, da die Ausbreitung von WLAN-Netzwerken an

Universitäten und Fachhochschulen sehr fortgeschritten ist. Ein Server sorgt dafür, dass sämtliche gespeicherte Fragen und Ergebnisse jederzeit von überall erstellt, bearbeitet oder abgerufen werden können.

3.2 Interviews mit potenziellen Nutzern

Zusätzlich zu den unter Punkt 3.1 beschriebenen Überlegungen wurden ein Student und ein Professor zu ihren Wünschen und Ideen interviewt.

3.2.1 Professor

Befragt wurde ein Professor, der im Studiengang „Angewandte Informatik“ an der Hochschule für Angewandte Wissenschaften Hamburg lehrt. Kernaussage war die Forderung eines intuitiven, einfach zu bedienenden Systems, welches wenig Einarbeitungsaufwand erfordert und eine schnelle Integration in das Vorlesungsmaterial ermöglicht.

3.2.2 Student

Befragt wurde ein Student des Studiengangs Volkswirtschaftslehre an der Universität Hamburg. Dieser hat das unter Punkt 3.1 beschriebene Clicktracker-System der Universität Hamburg bereits mehrfach in Vorlesungen zum Einsatz kommen sehen und selber benutzt. Positiv in Erinnerung blieben dem Studenten von der Nutzung folgende Punkte:

- Interaktive Inhalte binden die Studenten besser in die Vorlesungen ein und sind sehr anschaulich.
- Die direkte Abfrage des neu gelernten Wissens erleichtert den Lernprozess.

Negativ fiel der Zeitaufwand auf:

- Der Aufbau dauert durch die Ausgabe der Endgeräte sehr lange.
- Jeder Clicker muss einmalig am Anfang der Vorlesung am System angemeldet werden, was viel Zeit in Anspruch nimmt.
- Abgabe der Hardware-Clicker dauert sehr lange.

Folgende Wünsche wurden hinsichtlich eines neuen Clicktracker-Systems geäußert:

- Eigene Antworten sollen nachträglich auswertbar sein.
- Fragen sollen als Lernhilfe erneut beantwortbar sein.

3.2.3 Fazit

Um die Wünsche der Professoren umzusetzen, muss sich mehr auf die Kernfunktion, als auf viele optionale Features konzentriert werden.

Viele Probleme, die von dem Studenten angesprochen wurden, sind bereits unter Punkt 3.1 erwähnt worden. Das Verwenden von Smartphones und anderen mobilen Geräten, sowie die zentrale Speicherung aller Daten auf einem Server, löst diese Probleme.

3.3 Anforderungen

Basierend auf den Erkenntnissen aus diesem Kapitel werden funktionale und nicht-funktionale Anforderungen beschrieben.

3.3.1 Funktionale Anforderungen

Hier folgen die funktionalen Anforderungen:

Rollen

A01 Es sollen Benutzer definierbar sein. Benutzer können Administratoren, Professoren, Studenten oder anonyme Studenten sein.

A02 Jedem Benutzer ist eine E-Mail-Adresse und ein Passwort zugeordnet.

A03 Benutzer sind eindeutig über eine E-Mail-Adresse identifizierbar.

A04 Es ist kein Benutzerkonto nötig, um an Inhalten teilzunehmen. Studenten können sich freiwillig ein Konto anlegen, welches die Möglichkeit bietet, eine Statistik [A17] über die eigenen Leistungen abzurufen.

Vorlesungen und Vorlesungstermine

A05 Es sollen Vorlesungen von einem Professor angelegt werden können. Vorlesungen bestehen aus Vorlesungsterminen.

A06 Vorlesungen sind durch eindeutige Nummern identifizierbar. Sie bestehen aus der Abkürzung VL, gefolgt von einer fortlaufenden Nummer; Beispiel: „VL-15“.

A07 Für jede Vorlesung können Vorlesungstermine angelegt werden. Vorlesungstermine bestehen aus Aufgaben [A11] und einer Statistik [A17].

A08 Vorlesungstermine sind durch eindeutige Nummern identifizierbar. Sie bestehen aus der Abkürzung VT, gefolgt von einer fortlaufenden Nummer; Beispiel: „VT-15“.

A09 Ein Vorlesungstermin wird durch folgende Attribute beschrieben:

a) Kurze Information über den Inhalt des Termins.

b) Datum des Vorlesungstermins (wird entsprechend angepasst, wenn der Vorlesungstermin aktiviert wird).

A10 Ein Vorlesungstermin soll am Anfang der Vorlesung vom Professor aktiviert werden können. Es soll daraufhin ein QR- und entsprechender Zahlencode erzeugt werden, über den sich die Clients der Studenten für diesen Termin anmelden können.

Aufgaben

A11 Aufgaben sind durch eindeutige Nummern identifizierbar. Sie bestehen aus der Abkürzung AG, gefolgt von einer fortlaufenden Nummer; Beispiel: „AG-15“.

A12 Eine Aufgabe wird durch folgende Attribute beschrieben:

a) Aufgabenbeschreibung.

b) Aufgabentyp (z.B. „Multiple-Choice“ oder „Rechnung“).

c) Antwort.

A13 Eine Aufgabe soll vom Professor während einer Vorlesung aktiviert werden können, damit die Clients der Studenten diese beantworten können. Sobald er diese wieder beendet, soll sie ausgewertet und die Ergebnisse gespeichert werden [A17].

A14 Aktivierte Aufgaben sollen von allen Clients beantwortbar sein, die sich für den entsprechenden Vorlesungstermin angemeldet haben.

A15 Gespeicherte Ergebnisse für Studenten-Benutzeraccounts sollen auswertbar sein.

A16 Eine Aufgabe kann eine Zeitangabe beinhalten, nach der die Aufgabe automatisch beendet und ausgewertet wird.

Statistik

A17 Es wird für jeden Benutzer [A01], jede Aufgabe [A11] und jeden Vorlesungstermin [A07] eine Statistik geführt.

A18 Eine Statistik ist durch eine eindeutige Nummer identifizierbar. Sie besteht aus der Abkürzung ST, gefolgt von einer fortlaufenden Nummer; Beispiel: „ST-15“.

A19 Nach dem Beenden einer Aufgabe wird diese ausgewertet und eine Statistik für sie erstellt. Die Statistik aller Benutzer, die diese Aufgabe beantwortet haben, sowie die Statistik des Vorlesungstermins, zu dem die Vorlesung gehört, werden in der Folge aktualisiert.

A20 Die Statistik eines Benutzers soll im Studenten-Client visualisierbar sein.

A21 Die Statistiken für Aufgaben und Vorlesungstermine sollen im Professor-Client visualisierbar sein.

3.3.2 Nicht-funktionale Anforderungen

A22 Weitere Clients müssen für das System ohne Änderung der Server-Logik entwickelbar sein.

A23 Der Server muss bis zu 100 Antworten für eine Aufgabe innerhalb von 10 Sekunden für eine Visualisierung auswerten können.

4 Spezifikation

In diesem Kapitel wird das Clicktracker-System basierend auf den Anforderungen (siehe 3.3) spezifiziert.

4.1 Fachliches Datenmodell

Den Anforderungen entsprechend ist ein fachliches Datenmodell entstanden (Abbildung 4.1). Die beiden zentralen Klassen sind die Benutzer-Klasse und die Aufgaben-Klasse. Benutzer besitzen einen Login und gehören zu einer Benutzergruppe. Entsprechend ihrer Rolle und deren Berechtigung können sie Aufgaben beantworten oder erstellen, sowie diese Vorlesungsterminen zuordnen. Da die Teilnahme ebenfalls anonym ermöglicht werden soll, gibt es eine Rolle „Anonym“.

Aufgaben besitzen eine Aufgabenbeschreibung und Antworten. Zu jeder Aufgabe wird beim Anlegen eine Antwort mit der richtigen Lösung erzeugt. Die Überprüfung auf richtige Beantwortung einer Frage wird so vereinfacht. Antworten werden Benutzern zugeordnet, damit eine Benutzerstatistik erstellbar ist.

Vorlesungstermine sind Teile von ganzen Vorlesungen, welche dem Umfang eines Semesters entsprechen. Weiterhin können Vorlesungstermine von Benutzern belegt werden.

Jede Aufgabe besitzt eine Statistik. Es sollen während der Vorlesungstermine gestellte Fragen auswertbar und visualisierbar sein. Entsprechend der Aufgabenstatistik gibt es auch für Vorlesungstermine eine Statistik. Das Attribut für die durchschnittliche Dauer einer Antwort ist ein prozentualer Wert, der durch die zum Antworten benötigte Zeit im Verhältnis zur festgelegten Maximalzeit berechnet wird.

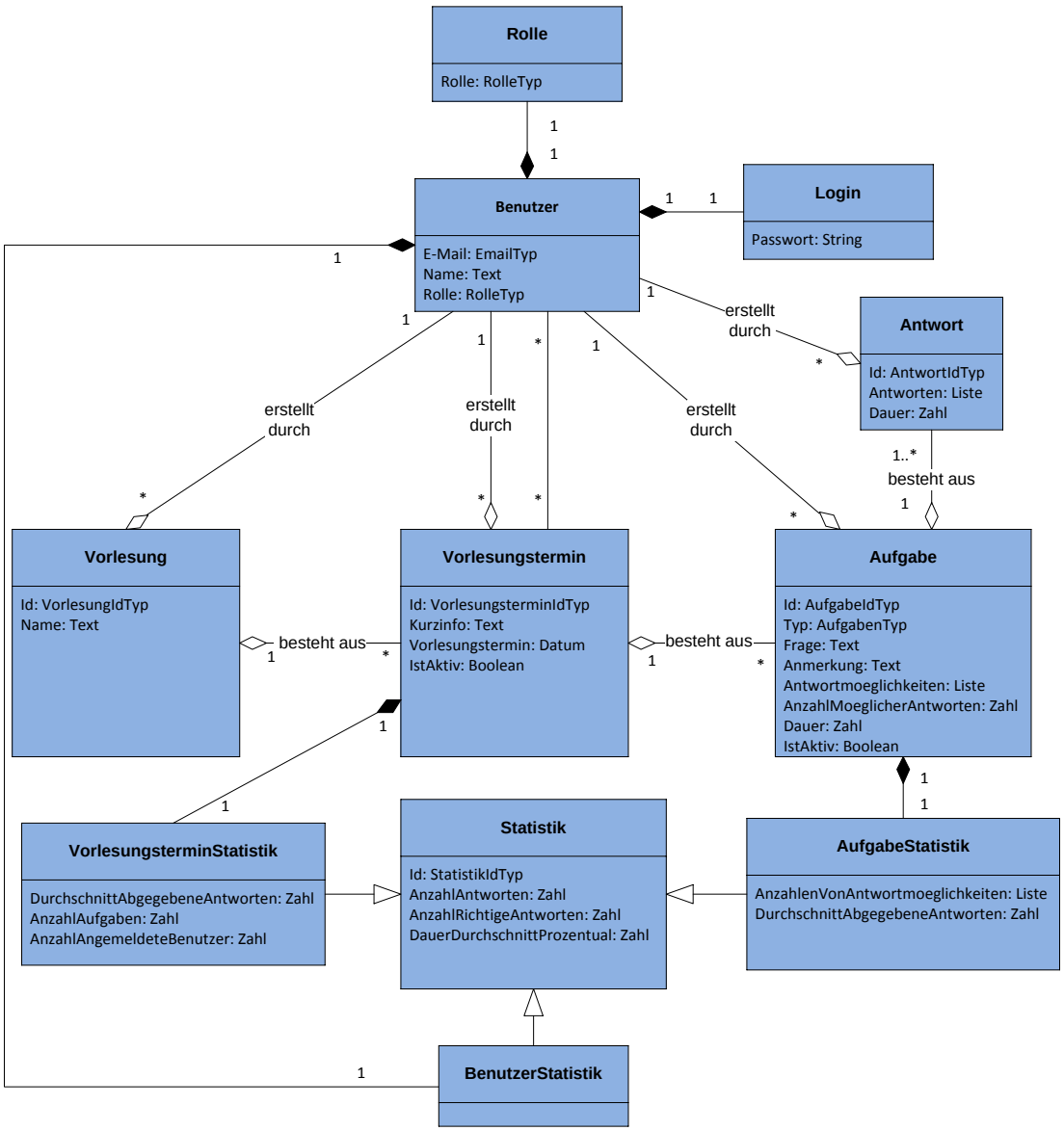

Abbildung 4.1: Fachliches Datenmodell

4.2 Beschreibung der fachlichen Datentypen

Hier folgt eine genaue Beschreibung der fachlichen Datentypen des fachlichen Datenmodells.

AntwortIdTyp

Datentyp	AntwortIdTyp
Beschreibung	Identifiziert eine Antwort eindeutig
Wertebereich	AW-\d+
GUI-Darstellung	Darstellung des Wertes

AufgabeIdTyp

Datentyp	AufgabeIdTyp
Beschreibung	Identifiziert eine Aufgabe eindeutig
Wertebereich	AG-\d+
GUI-Darstellung	Darstellung des Wertes

AufgabeTyp

Datentyp	AufgabenTyp
Beschreibung	Repräsentiert den Aufgabentyp einer Aufgabe
Wertebereich	{„MULTIPLE_CHOICE“, „RECHNUNG“, „WAHR_FALSCH“, „JA_NEIN“}
GUI-Darstellung	Darstellung des Wertes

EmailTyp

Datentyp	EmailTyp
Beschreibung	Repräsentiert eine E-Mail-Adresse nach RFC-2822
Wertebereich	Siehe RFC-2822
GUI-Darstellung	<Local-Part>@<Domain>

RolleTyp

Datentyp	RolleTyp
Beschreibung	Repräsentiert eine Rolle für einen Benutzer
Wertebereich	{„PROFESSOR“, „STUDENT“, „ANONYM“, „ADMINISTRATOR“}
GUI-Darstellung	Darstellung des Wertes

StatistikIdTyp

Datentyp	StatistikIdTyp
Beschreibung	Identifiziert eine Statistik eindeutig
Wertebereich	ST-\d+
GUI-Darstellung	Darstellung des Wertes

VorlesungIdTyp

Datentyp	VorlesungIdTyp
Beschreibung	Identifiziert eine Vorlesung eindeutig
Wertebereich	VL-\d+
GUI-Darstellung	Darstellung des Wertes

VorlesungsterminIdTyp

Datentyp	VorlesungsterminIdTyp
Beschreibung	Identifiziert einen Vorlesungstermin eindeutig
Wertebereich	VT-\d+
GUI-Darstellung	Darstellung des Wertes

4.3 Anwendungsfälle

Drei beispielhafte Anwendungsfälle werden unter diesem Punkt spezifiziert. Für einen generellen Überblick und um die Beziehung zwischen den Akteuren und beispielhaften Anwendungsfällen zu verdeutlichen, ist in Abbildung 4.2 ein Anwendungsfalldiagramm dargestellt.

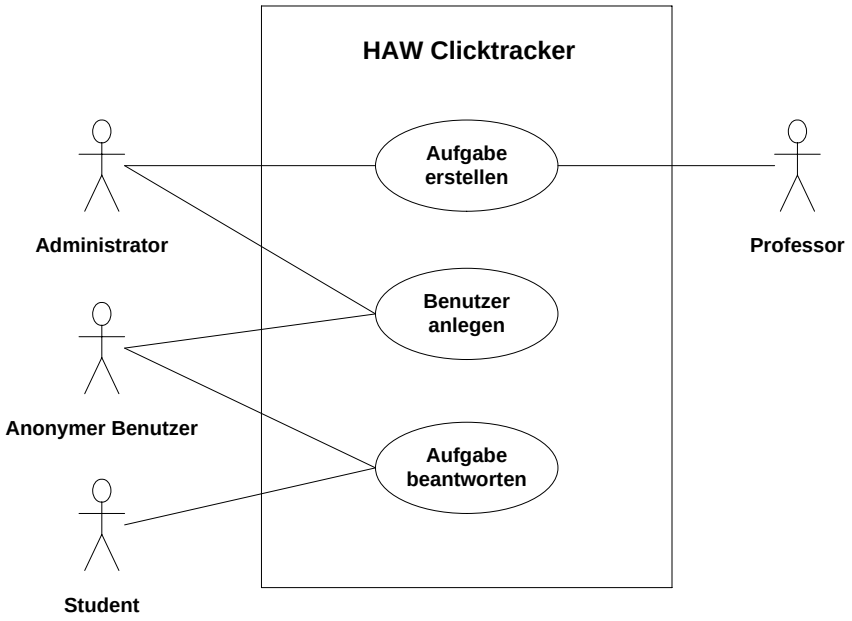

Abbildung 4.2: Anwendungsfalldiagramm für drei beispielhafte Anwendungsfälle

4.3.1 Benutzer anlegen

Titel	Benutzer anlegen
Akteur	Benutzer
Ziel	Benutzeraccount ist im System registriert
Auslöser	Benutzer möchte Benutzeraccount anlegen
Vorbedingung	Benutzer hat noch keinen Benutzeraccount
Nachbedingung	Benutzeraccount für Benutzer ist angelegt
Erfolgszenario	
1 Der Benutzer ruft die Registrierung auf. 2 Der Benutzer füllt das Formular der Registrierung aus. 3 Der Benutzer schickt das Formular ab. 4 Das System überprüft die Eingaben des Benutzers. 5 Das System legt einen neuen Benutzer an und bestätigt dies.	
Erweiterungen	
Fehlerfälle	
4a E-Mail-Adresse entspricht nicht der Vorgabe: System meldet Fehler und fordert zur Korrektur auf. 4b E-Mail-Adresse ist bereits vergeben: System meldet Fehler und fordert zur Korrektur auf. 4c Ein Feld des Formulars ist nicht ausgefüllt: System meldet Fehler und fordert zur Korrektur auf.	
Häufigkeit	Einmalig für eine Emailadresse
Anforderungen	A01 - A04

Tabelle 4.1: Anwendungsfall: „Neuen Benutzer anlegen“

4.3.2 Aufgabe erstellen

Titel	Aufgabe erstellen
Akteur	Benutzer
Ziel	Eine Aufgabe ist erstellt
Auslöser	Ein Benutzer möchte eine Aufgabe erstellen
Vorbedingung	Benutzer ist am System angemeldet und gehört zur Benutzergruppe der Professoren
Nachbedingung	Aufgabe ist angelegt
Erfolgszenario	
1 Der Benutzer ruft den Dialog „Neue Aufgabe“ auf. 2 Der Benutzer wählt den Aufgabentyp aus. 3 Der Benutzer füllt das Formular für die entsprechende Aufgabe aus. 4 Der Benutzer schickt das Formular ab. 5 Das System überprüft die Eingaben des Benutzers. 6 Das System bestätigt die Erstellung der Aufgabe.	
Erweiterungen	
Fehlerfälle	
5a Ein notwendiges Feld des Formulars ist nicht ausgefüllt: System meldet Fehler und fordert zur Korrektur auf.	
Häufigkeit	
Anforderungen	A11 - A16

Tabelle 4.2: Anwendungsfall: „Neue Aufgabe erstellen“

4.3.3 Aufgabe beantworten

Titel	Aufgabe beantworten
Akteur	Benutzer
Ziel	Aktive Aufgabe beantwortet
Auslöser	Ein Benutzer möchte eine aktive Aufgabe beantworten
Vorbedingung	Benutzer ist für Vorlesungstermin dieser Aufgabe angemeldet
Nachbedingung	Aufgabe ist von Benutzer beantwortet
Erfolgszenario	
1 Der Benutzer ruft den Dialog zum Beantworten einer Aufgabe auf. 2 Das System zeigt das Antwort-Formular für die aktive Aufgabe. 3 Der Benutzer füllt das Formular für die Antworten aus. 4 Der Benutzer schickt das Formular ab. 5 Das System überprüft die Eingaben des Benutzers. 6 Das System bestätigt das Beantworten der Aufgabe.	
Erweiterungen	
3a Der Benutzer bricht das Beantworten der Frage ab. 3b Das System schickt das Auslassen der Frage ab.	
Fehlerfälle	
5a Ein notwendiges Feld des Formulars ist nicht ausgefüllt: System meldet Fehler und fordert zur Korrektur auf.	
Häufigkeit	
Anforderungen	A11 - A16

Tabelle 4.3: Anwendungsfall: „Aktive Aufgabe beantworten“

4.4 Dialoge

In diesem Abschnitt folgen erste Mockups, die einen Eindruck der jeweiligen Anwendung des Systems vermitteln. Sie besitzen keinerlei Funktionalität und stellen nicht den finalen Stand der grafischen Oberfläche dar. Trotzdem sind sie wichtig, um einen Eindruck davon zu gewinnen, worauf Wert gelegt werden muss, um das System intuitiv und einfach verwendbar zu machen.

4.4.1 Studenten-Anwendung

Bei der Studentenanwendung wird Wert auf eine simple und intuitive Oberfläche gelegt. Der Student soll sich mit der Aufgabe fachlich auseinandersetzen können, ohne dabei Probleme mit der Bedienung zu haben. Es wird sich auf die nötigsten Bedienelemente beschränkt, die zur Beantwortung nötig sind. Weiterführende Informationen werden bei Bedarf durch die Professoren-Anwendung dargestellt. Abbildung 4.3 zeigt einen Mockup einer minimalen Oberfläche zum Beantworten einer Aufgabe.

Abbildung 4.3: Mockup einer Oberfläche zum Beantworten einer Frage

Die Statistik eines Benutzers ist bisher so spezifiziert, dass mit der Anzahl der richtigen und falschen Antworten, der Anzahl der gesamten gegebenen Antworten und der im Durchschnitt gebrauchten Zeit, vier darzustellende Werte gegeben sind. Hier macht ein Kreisdiagramm Sinn, um das Verhältnis zwischen richtigen und falschen Antworten zu zeigen. Die anderen beiden Werte werden textuell dargestellt. Ein Mockup für die Statistik-Oberfläche ist in Abbildung 4.4 dargestellt.

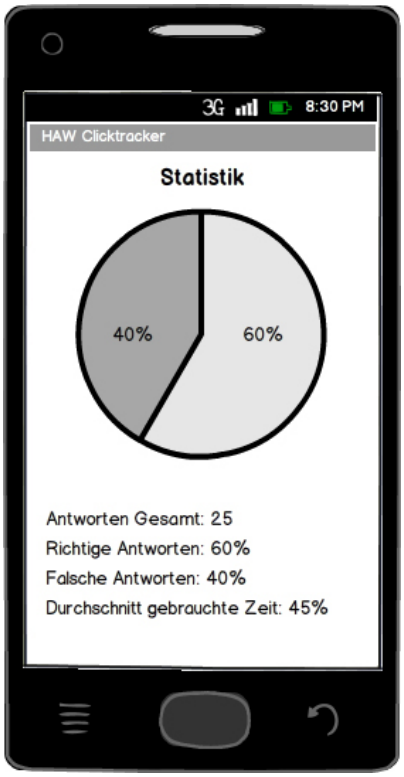

Abbildung 4.4: Mockup einer Oberfläche zum Anzeigen der Benutzerstatistik

4.4.2 Professor-Anwendung

Bei der Professor-Anwendung des Systems liegt der Fokus besonders auf einem schnellen und einfachen Ablauf. Dies betrifft in erster Linie die Quantität der Einsätze des Clicktracker-Systems, da bei großem Mehraufwand während der Vorlesungsvorbereitung das Verhältnis von Aufwand zu Nutzen nicht mehr stimmt.

Abbildung 4.5 zeigt ein Mockup der Verwaltungs-Oberfläche. Dort sind übersichtlich alle Vorlesungen, sowie deren zugehörige Vorlesungstermine und entsprechend Aufgaben

dargestellt. Die rechte Bildschirmseite zeigt beispielhaft das Formular zum Anlegen eines Vorlesungstermins.

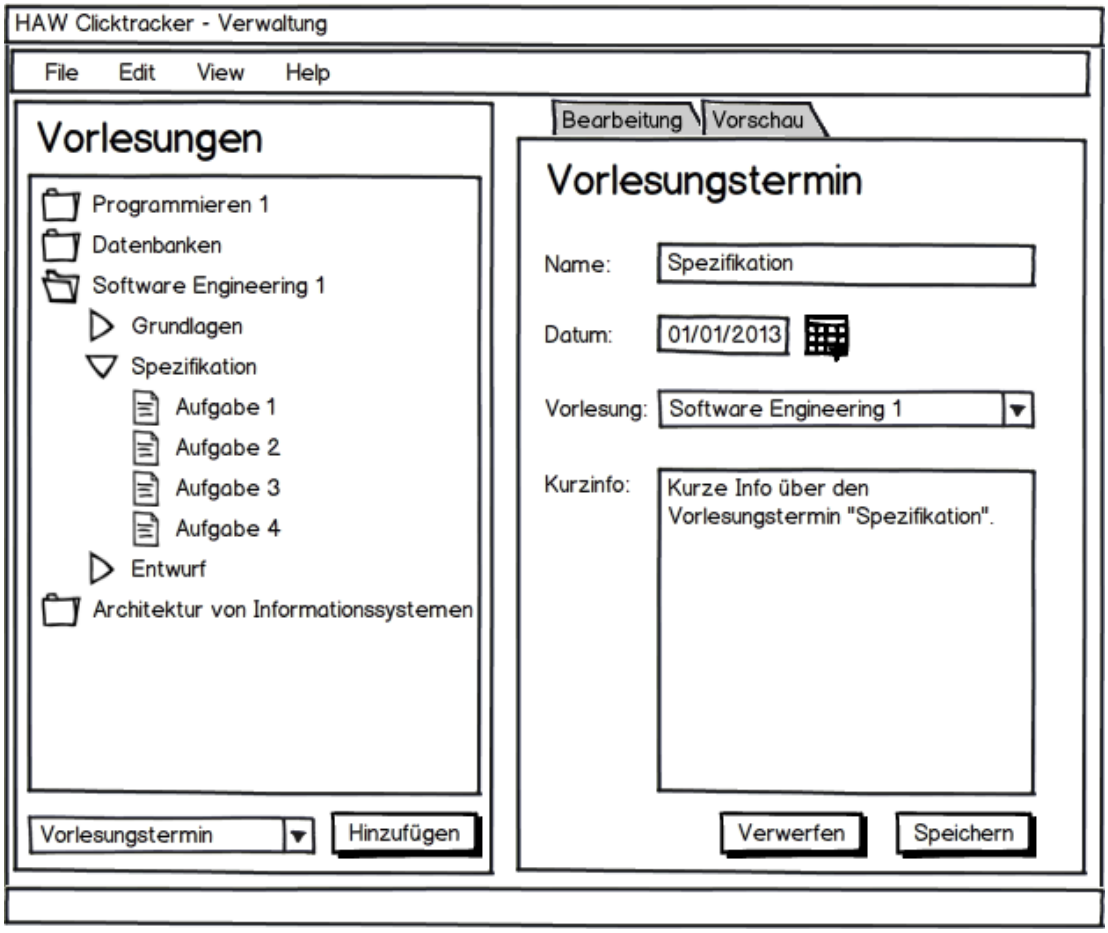

Abbildung 4.5: Mockup der Verwaltungsoberfläche der Professor-Anwendung

5 Entwurf

In diesem Kapitel wird die Architektur des Systems entwickelt und beschrieben. Im ersten Schritt werden Komponenten gesucht, im zweiten der Webservice beschrieben und im letzten Schritt die Architektur des Systems erläutert.

5.1 Komponenten

Eine Komponente wird nach Siedersleben durch fünf Merkmale beschrieben (Siedersleben, 2004):

1 Sie exportiert garantierte Schnittstellen.

2 Sie kann Schnittstellen importieren, um dessen Methoden zu nutzen.

3 Sie ist eine Black Box, deren Implementierung für andere Komponenten des Systems nicht ersichtlich ist. Dies hat den Vorteil, dass Komponenten austauschbar sind, solange sie dieselben Schnittstellen anbieten.

4 Sie trifft keine Annahmen über die Umgebung, da sie so wiederverwendbar ist.

5 Komponenten können andere Komponenten kapseln und verwenden.

Diesen Merkmalen entsprechend werden im Folgenden Komponenten und deren Beziehungen untereinander aus dem fachlichen Datenmodell abgeleitet. Des Weiteren werden am Beispiel einer dieser Komponenten die Außen- und Innenansicht beschrieben.

5.1.1 Komponentenschnitt

In Darstellung 4.1 ist die Abbildung des fachlichen Datenmodells auf Komponenten gezeigt.

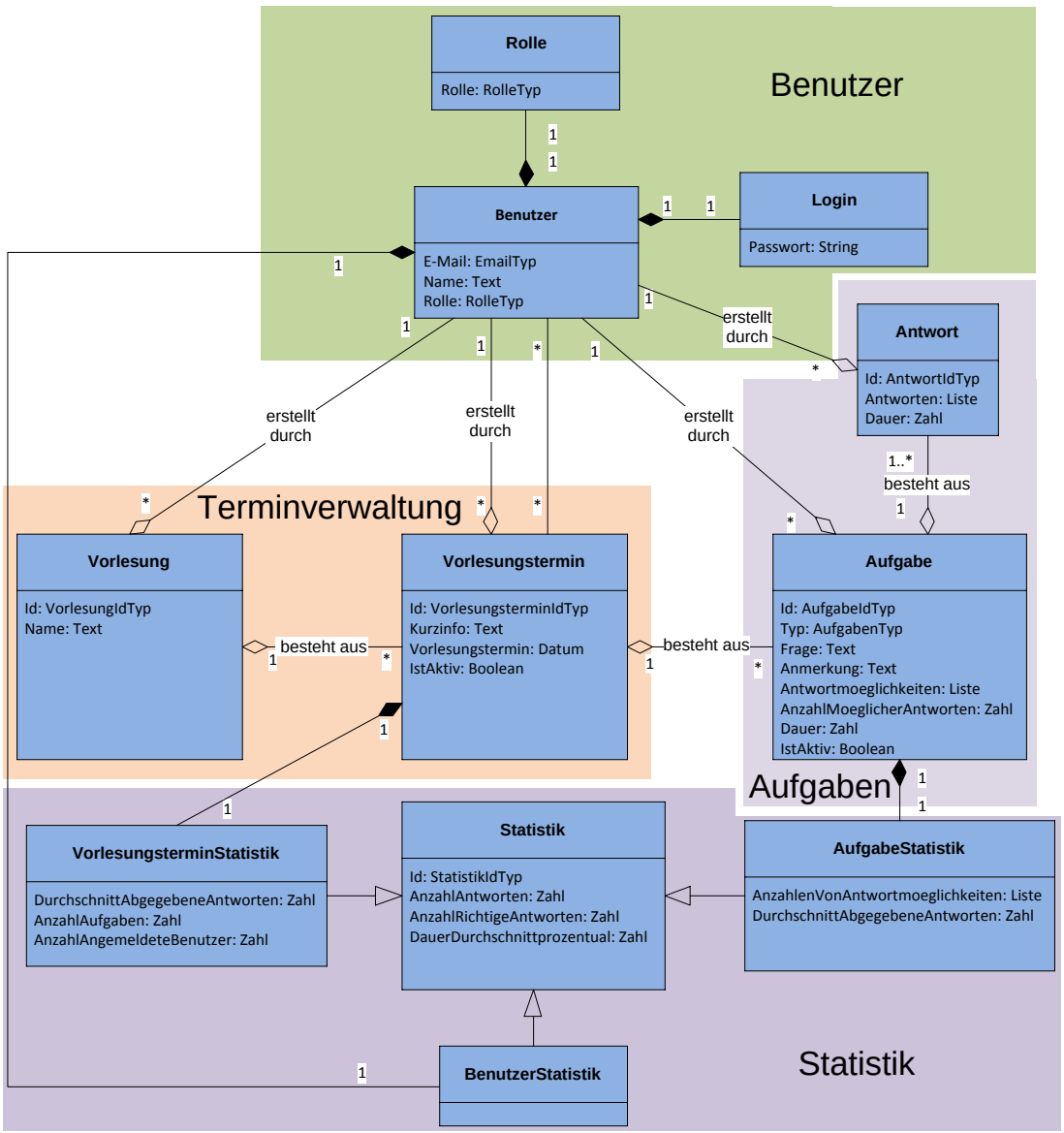

Abbildung 5.1: Abbildung des fachlichen Datenmodells (Abbildung 4.1) auf Komponenten

Es sind vier Komponenten entstanden:

1 BenutzerKomponente

Verwaltet die Benutzer.

2 TerminverwaltungKomponente

Verwaltet die Veranstaltungen.

3 AufgabenKomponente

Verwaltet die Aufgaben und deren Antworten.

4 StatistikKomponente

Verwaltet die Statistik für Benutzer, Aufgaben und Vorlesungstermine.

5.1.2 Komponentendiagramm

In Abbildung 5.2 ist das Komponentendiagramm, welches unter 5.1.1 entwickelt wurde, dargestellt. Es zeigt die Komponenten der Server-Anwendung in der Übersicht, sowie deren bereitgestellte und verwendete Schnittstellen.

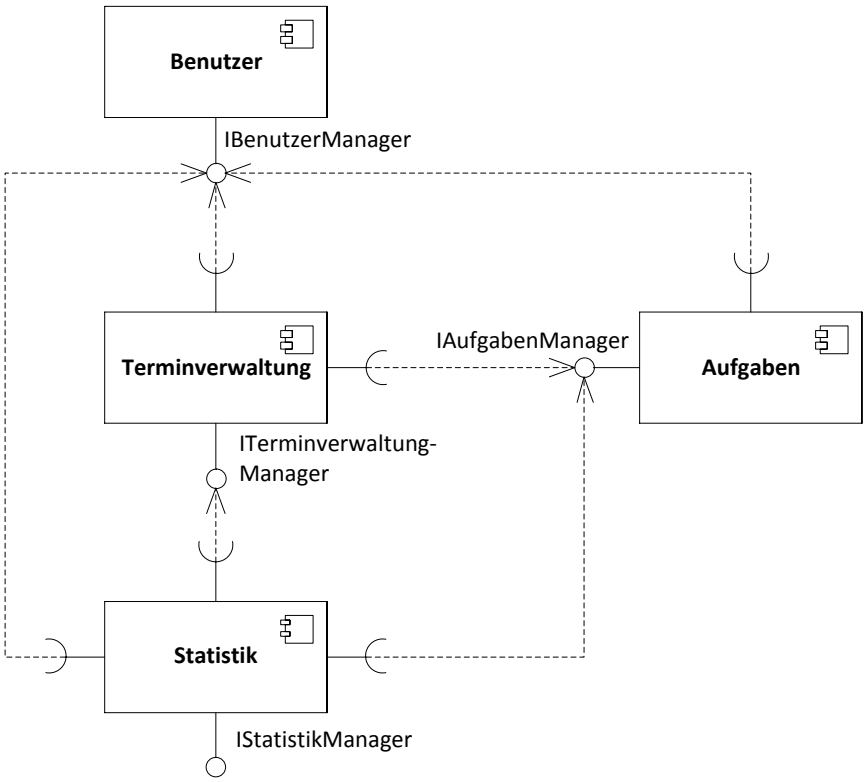

Abbildung 5.2: Komponentendiagramm

5.1.3 Außensicht

Die im Komponentendiagramm dargestellten angebotenen Schnittstellen der Komponenten müssen entworfen und beschrieben werden. Für diese Dokumentation gibt es keinen Standard. In dieser Arbeit wurde die Beschreibung durch Javadoc-konforme Kommentare vorgenommen.

Dies wird anhand von zwei Methoden verschiedener Schnittstellen und Komponenten verdeutlicht. Hierfür werden wieder die bereits vorgestellten beispielhaften Anwendungsfälle (siehe 4.3) als Beispiel genommen. In Listing 5.1 ist die Definition der „erzeugeBenutzer“-Methode der Schnittstelle „IBenutzerManager“ der BenutzerKomponente mitsamt der Dokumentation gezeigt. In Listing 5.2 ist die Definition der „erzeugeAntwort“-Methode der Schnittstelle „IAufgabeManager“ der AufgabeKomponente gezeigt.

```
1  /**
2   * Erzeugt einen neuen Benutzer
3   *
4   * @param email Eindeutige Email fuer den neuen Benutzer
5   * @param passwort Passwort fuer neuen Benutzer
6   * @param Rolle Rolle fuer den neuen Benutzer
7   * @return Liefert das Transportobjekt des neu erzeugte Benutzerobjekt
8   * zurueck. Sollte die Erzeugung fehlschlagen wird null zurueckgeliefert
9   */
10 BenutzerTyp erzeugeBenutzer(String email, String passwort, RolleTypEnum
       rolleTyp);
```

Listing 5.1: Ausschnitt der Dokumentation der Schnittstelle „IBenutzerManager“ der BenutzerKomponente

```
/**
 * Erzeugt eine Antwort
 *
 * @param dauer Dauer der Antwort
 * @param aufgabeZuAntwortId Aufgabe, fuer die die Antwort gilt. Null, falls
 * eine initiale richtige Antwort ohne bereits vorhandene Aufgabe erstellt
 * werden soll
 * @param antworten Antworten fuer die Aufgabe
 * @param erstellerId Eindeutige Id des Erstellers der Antwort
 * @return Das Transportobjekt der erzeugten Antwort. Sollte die Aufgabe
 * fuer die diese Antwort gilt nicht mehr aktiv sein, wird null
 * zurueckgegeben
 */
AntwortTyp erzeugeAntwort(int dauer, String aufgabeZuAntwortId, List<Integer
    > antworten, String erstellerId);
```

Listing 5.2: Ausschnitt der Dokumentation der Schnittstelle „IAufgabeManager“ der AufgabeKomponente

5.1.4 Innensicht

Nach der Außensicht folgt hier die Innensicht einer Komponente. Als Beispiel wurde wieder die Benutzerkomponente gewählt, welche die Verwaltung der Benutzer kapselt. In Abbildung 5.3 ist die Innenansicht die Komponente gezeigt. Ein Benutzeranwendungsfall implementiert die Logik der Komponente und interagiert entsprechend mit den Entitäten und deren Verwaltern.

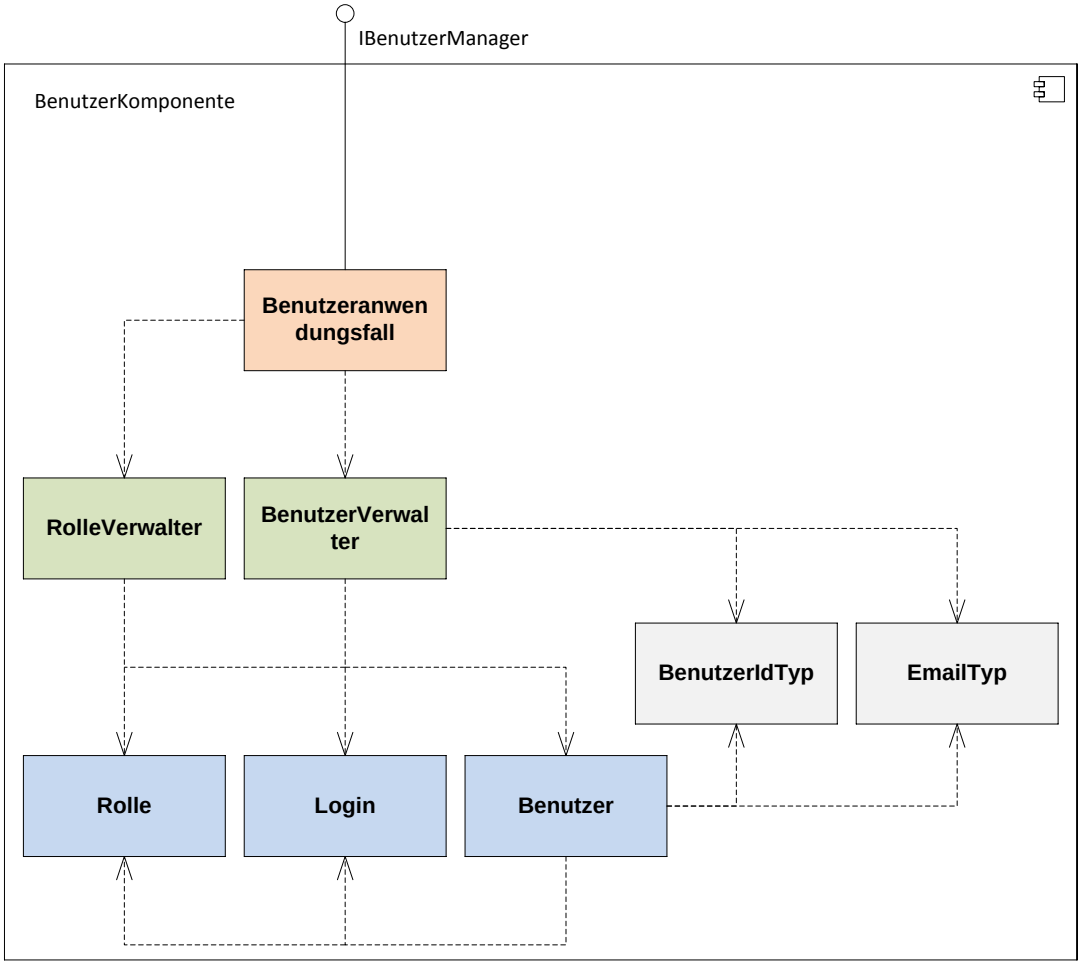

Abbildung 5.3: Innenansicht der BenutzerKomponente

5.1.5 Verbindung der Komponenten (Konfiguration)

Nachdem das System durch die Komponenten in geschlossene Einheiten geteilt wurde, müssen diese noch verbunden werden. Hier kommt mit „Dependency Injection“ ein bekanntes Entwurfsmuster zum Tragen. Dies ist sinnvoll, da die Komponenten sich so nicht um die Erzeugung von Implementierungen der importierten Schnittstellen kümmern müssen und die Abhängigkeiten zwischen den Komponenten weiter minimal bleiben.

Martin Fowler beschreibt dieses Muster (Fowler, 2004) und visualisiert die Problemstellung durch Diagramme. Der naheliegendste Ansatz zum Import von Schnittstellen ist in Abbildung

5.4 dargestellt. Es wird sowohl das angebotene Interface importiert als auch die konkrete Implementierung erzeugt und verlinkt. Ein Austausch der Komponente durch eine andere, die die gleiche Schnittstelle exportiert, wäre so nicht ohne Weiteres möglich.

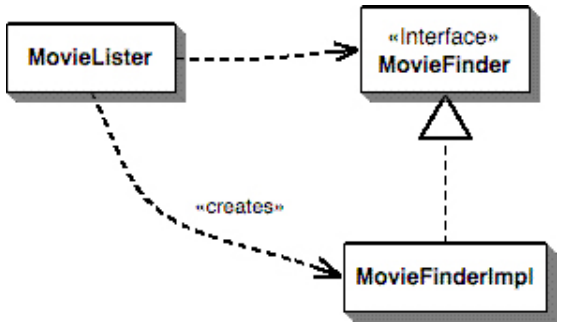

Abbildung 5.4: Erzeugung von Abhängigkeiten zwischen Komponenten ohne „Dependency Injection“ (Fowler, 2004)

Um diese letzte große Abhängigkeit aufzuheben, kommt das „Dependency Injection“-Muster zum Tragen. Die Komponente importiert nur noch das Interface der angebotenen Schnittstelle. Die konkrete Implementierung dieser Schnittstelle wird durch einen Assembler erzeugt, der diese dann der Komponte, welche die Schnittstelle importiert, injiziert. Damit gibt es mit dem Assembler nur noch eine zentrale Stelle, die für die Erzeugung der konkreten Implementierungen zuständig ist. Abgebildet ist dieser Ansatz in Abbildung 5.5.

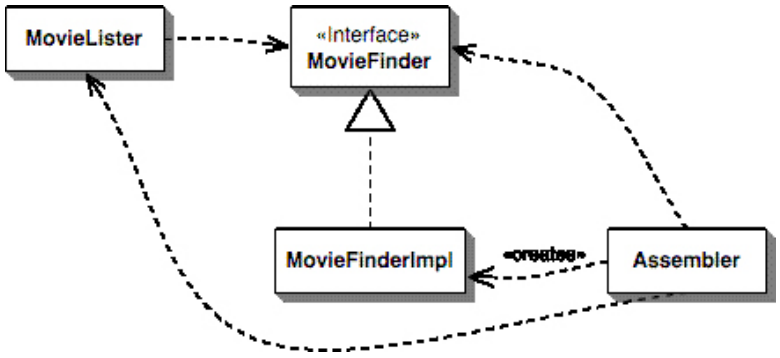

Abbildung 5.5: Erzeugung von Abhängigkeiten zwischen Komponenten mit „Dependency Injection“ (Fowler, 2004)

Für das Injizieren in die Komponenten gibt es nach Fowler (Fowler, 2004) drei Ansätze. Konstruktor-Injizierung, Setter-Injizierung und Interface-Injizierung. In dieser Arbeit wird die Injizierung über Konstruktoren verwendet. Dies ermöglicht ein einfaches Testen der Komponenten, da Mockups für die importierten Schnittstellen einfach injiziert werden können.

5.2 Webservice

In den Grundlagen wurden mit SOAP und REST zwei verbreitete Ansätze vorgestellt (siehe 2.3), welche im Folgenden verglichen werden.

Der größte Vorteil von REST ist die Interoperabilität, da es auf HTTP aufsetzt. REST bietet eine einfach zu erweiternde Schnittstelle, durch welche nachträglich verschiedene Clients für Studenten einfacher in das System integriert werden können. Der in dieser Arbeit entwickelte Client für Android muss dabei nicht angepasst werden. Bei SOAP wäre dies nur über den Umweg möglich, die geänderte Schnittstelle als zusätzliche Schnittstelle zu implementieren.

Im Gegensatz zu SOAP bietet REST die Möglichkeit andere Repräsentationen als XML zu verwenden. Dies wird im nächsten Abschnitt 5.2.1 genauer betrachtet.

Verteilte Transaktionen oder schwergewichtige, lange dauernde Operationen, welche für SOAP sprechen würden, werden in dieser Anwendung nicht benötigt.

Zusammenfassend spricht vieles für einen REST-Webservice in dieser Arbeit. Das bedeutet nicht, dass SOAP für diese Anwendung ungeeignet ist, jedoch ist die Entscheidung aufgrund der zuvor genannten Punkte auf REST gefallen.

5.2.1 Datenaustauschformat

In Verbindung mit der Entscheidung für REST, bietet sich die Möglichkeit, JSON als Alternative zu XML zu verwenden. Relevante Faktoren sind bei dieser Entscheidung sowohl die Schnelligkeit, als auch die Ressourcen-Aufwendung. Dies ist vor allem für den mobilen Client für Studenten wichtig. Eine Anwendung, die nicht performant ist, sorgt für eine schlechte Benutzererfahrung, was sich letztendlich negativ auf die Beteiligung der Studenten niederschlägt. Weiterhin spielt bei einem mobilen Client die Akkulaufzeit eine große Rolle. Stärkere Verwendung der Hardware eines mobilen Endgerätes schlägt sich in der Akkulaufzeit nieder.

Konkret getestet wurde der Unterschied zwischen JSON und XML in einer Fallstudie der Montana State University (Nurseitov u. a., 2009). Es wurden Testfälle entwickelt, bei denen die Übertragungsgeschwindigkeit und die Ressourcen-Aufwendung getestet wurden. Diese zeigen, dass je nach Anzahl der zu sendenden Objekte JSON gegenüber XML 28- bis 57-mal schneller ist. Im Gegensatz dazu ist die CPU-Belastung durch JSON höher.

Um das System so performant wie möglich zu entwerfen, ist JSON das Format, welches verwendet wird. Der Geschwindigkeitsvorteil ist in diesem Anwendungsgebiet deutlich wichtiger einzuschätzen als der Nachteil durch die höhere CPU-Belastung.

5.2.2 Kommunikationsprotokoll

Im Zuge der Entwurfsentscheidung für JSON wird ein Protokoll für den Austausch von Daten entworfen.

JSON besteht aus Objekten, die durch Schlüssel/Wert-Paare abgebildet werden. Der Schlüssel ist dabei eine eindeutige Zeichenkette. Der Wert kann eine Zeichenkette, eine Zahl, ein Array, ein weiter verschachteltes Objekt sein, oder aus einem der Ausdrücke „true“, „false“ oder „null“ bestehen.

Das folgende Beispiel zeigt das Protokoll für die Übermittlung eines Vorlesungstermins von der Server-Anwendung zu der Professor-Anwendung.

Schlüssel	Wert
vorlesungsterminId	<Id des Vorlesungstermins> [A08]
kurzinfo	<Kurze Info zum Vorlesungstermin> [A09a]
vorlesungsterminDatum	<Datum des Vorlesungstermins> [A09b]
istAktiv	<Status des Vorlesungstermins (true oder false)> [A10]
vorlesungId	<Id der Vorlesung, zu der der Termin gehört> [A06]
benutzerIds	<Ids der Benutzer, die für diesen Vorlesungstermin angemeldet sind>
erstellerId	<Id des Erstellers des Vorlesungstermins> [A02]
aufgaben	<Liste der Ids der Aufgaben, die dem Termin zugeordnet sind> [A07]

Tabelle 5.1: Kommunikationsprotokoll für die Übermittlung eines Vorlesungstermins

Eine beispielhafte JSON-Nachricht, die diesem Protokoll entspricht, ist in Listing 5.3 dargestellt.

```
{
  "vorlesungsterminId":"VT-1",
  "kurzinfo":"Einführung",
  "vorlesungsterminDatum":"1342476000000"
  "istAktiv":false,
  "vorlesungId":"VL-1",
  "benutzerIds":["email@student1.de","email@student2.de"],
  "erstellerId":"email@professor.de",
  "aufgaben":["AG-1","AG-2"]
}
```

Listing 5.3: JSON-Beispielnachricht

5.2.3 Ressourcen und Methoden

Die Ressourcen basieren auf den persistenten Entitäten des Systems (siehe 4.1). Für jede Entität gibt es einen Pfad, über den diese erreichbar ist. Dies ist bei dem System dieser Arbeit ein intuitiver Ansatz. Entsprechende Attribute der Entitäten werden über Subressourcen erreicht. Um die angebotenen Methoden abzubilden, werden die HTTP-Standard-Verben GET, POST, PUT sowie DELETE verwendet. In Tabelle 5.2 ist dieses Design beispielhaft für einen Teil der Benutzer-Entität und entsprechender Ressource gezeigt.

Pfad	Verb	Beschreibung
/benutzer	POST	Erzeugt einen Benutzer
/benutzer/{benutzerid}	GET	Liefert den Benutzer mit der Id „benutzerid“
/benutzer/{benutzerid}	PUT	Aktualisiert den Benutzer mit der Id „benutzerid“. Gibt es keinen Benutzer mit dieser Id, wird dieser angelegt.
/benutzer/{benutzerid}	DELETE	Loescht den Benutzer mit der Id „benutzerid“
/benutzer/{benutzerid} /vorlesungstermin	GET	Liefert den Vorlesungstermin eines Benutzers mit der Id „benutzerid“

Tabelle 5.2: Ressourcen-Design am Beispiel der Benutzerentität

Diese Ressourcen werden nicht mehr verändert, sobald sie einmal festgelegt sind, da dies die angebotene API für fremde Clients verändern würde.

Es wurde in diesem Entwurf nur die Grundstruktur der Ressourcen festgelegt. Alle benötigten Ressourcen wurden im Laufe der Implementierung hinzugefügt.

5.3 Architektur

Im Folgenden werden die verwendete Architektur und die Entwurfmuster näher beschrieben.

5.3.1 3-Schichten-Architektur

In Abbildung 5.6 ist die Architektur des Systems zu sehen. Diese basiert auf der „3-Schichten-Architektur“, welche das System grob in drei Schichten teilt:

1 Applikation

Diese Schicht repräsentiert die Oberfläche, das Front-End. Sie beinhaltet sowohl den ProfessorClient als auch den StudentenClient, welche für die Verwendung des Systems durch einen Benutzer bestimmt sind (siehe 1.2). Aufgabe dieser Schicht ist die Interaktion mit dem Benutzer.

2 Applikationslogik

Die Bezeichnung Applikationslogik umschreibt präzise die Funktion dieser Schicht. Sie ist für die Realisierung der Fachlichkeit des Systems verantwortlich.

3 Persistenz

Die Persistenz-Schicht dient der persistenten Speicherung der Daten des Systems und abstrahiert technische Details der physischen Datenbank.

Der Vorteil dieser Architektur ist die Vereinfachung einer Verteilung der Anwendung. Die Schichten können beliebig über verschiedene Knoten verteilt werden, da sie logisch unabhängig voneinander sind.

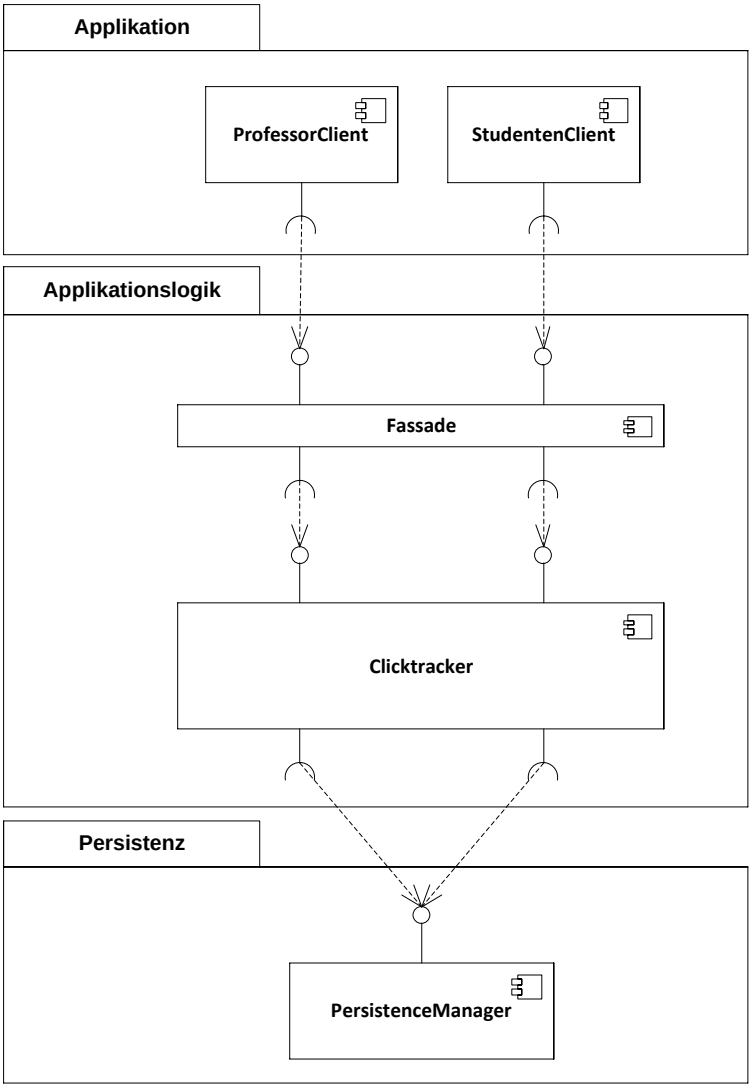

Abbildung 5.6: „3-Schichten-Architektur“

5.3.2 Entwurfsmuster „Fassade“

Zusätzlich zu der „3-Schichten-Architektur“ ist in der Abbildung 5.6 eine Fassade in der Applikationsschicht dargestellt. Dies folgt dem Entwurfsmuster „Fassade“, bei welchem es für alle Schnittstellen eines Subsystems eine einzige einheitliche Schnittstelle gibt. Dies löst Abhängigkeiten auf und vereinfacht in diesem System die Verteilung des Systems in Schichten.

5.3.3 Verteilung

Um die Verteilung der Schichten auf Knoten zu verdeutlichen, ist diese in Abbildung 5.7 als Verteilungsdiagramm dargestellt.

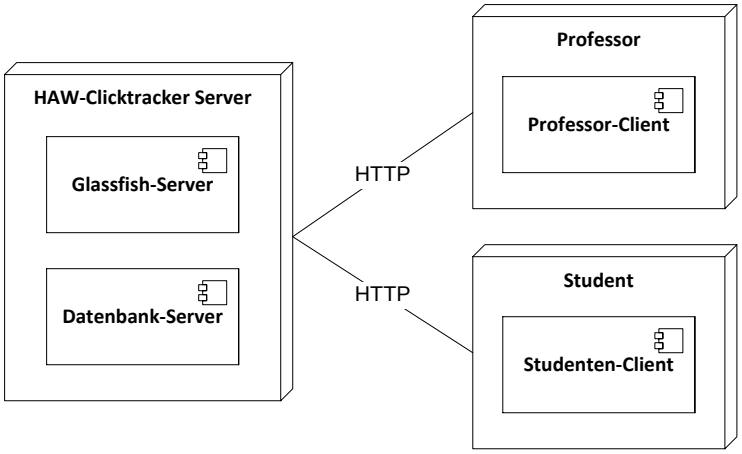

Abbildung 5.7: Die Verteilung des Systems als Verteilungsdiagramm dargestellt

5.3.4 TI-Architektur

In diesem Abschnitt wird die technische Infrastruktur des verteilten Systems näher erläutert. Dazu werden alle von Drittanbietern verwendeten Frameworks, Bibliotheken und anderweitige Softwarekomponenten festgelegt und deren Einsatzbereich beschrieben.

Software	Einsatzbereich
Java EE (Version 6)	Standardarchitektur für die Entwicklung der Server-Anwendung
Java SE (Version 6)	Standardarchitektur für die Entwicklung der Professor-Anwendung
GlassFish Server (Version 3.1.2)	Anwendungsserver für die Serveranwendung des Systems
Apache Derby (Version 10.8.1.2)	Datenbank für die Entwicklung
Apache Maven (Version 3.0.4)	Build-Management-Tool für die Server- und Professoranwendung
Hibernate (Version 4.1.2)	Persistenz-Framework
Jersey (Version 1.12)	Framework zur Entwicklung eines Webservice nach dem REST-Architekturstil (siehe 2.4)
JUnit (Version 3 & 4)	Framework zum Testen der Anwendungen des Systems. Version 4 kommt für die Server- und Professoranwendung zum Einsatz. Für die Androidanwendung kommt Version 3 mangels Unterstützung für Version 4 zum Einsatz
Mockito (Version 1.9.5-RC1)	Bibliothek zum Erstellen von Mock-Objekten in den Tests
Robotium (Version 3.3)	Test-Framework für Android. Ermöglicht automatisierte Oberflächentests.
Google Gson (Version 2.2.2)	Bibliothek zum Serialisieren und Deserialisieren von JSON-Objekten. Es wurde diese Bibliothek gewählt, da sie leichtgewichtig und klein ist (185 KB in Version 2.2.2)
ZXing (Version 2.0)	Bibliothek zum Erzeugen von QR-Codes
Google Guice (Version 3.0 without AOP)	Framework für Dependency Injection, welches in Android-Anwendungen verwendet werden kann
Google Cloud Messaging (Revision 3)	Push-Service für Android
JFreeChart (Version 1.0.0)	Bibliothek für Diagramme unter Java. Kommt in der Professor-Anwendung zum Einsatz
AChartEngine (Version 1.0.0)	Bibliothek für Diagramme in Android-Anwendungen

Tabelle 5.3: Verwendete Software von Drittanbietern

5.3.5 Sicherheit

Bei einer verteilten Anwendung stellt sich die Frage nach der Sicherheit für die Datenübertragung. Für diese Anwendung wird HTTPS in Verbindung mit Basic Auth verwendet. Stefan Tilkov nennt diese Option auch den 80%-Fall (Tilkov, 2011). Dies ist zwar nicht so sicher wie die Verbesserung durch ein verschlüsseltes Passwort (Digest Auth) oder OAuth, ist aber für den Rahmen, in dem diese Anwendung laufen wird, ausreichend.

6 Realisierung

In diesem Kapitel geht es um die Realisierung des Systems.

6.1 Umfang

Im Mittelpunkt der Implementation steht die Umsetzung des Grundsystems, mit welchem verschiedene Aufgabentypen und Arten der Visualisierung möglich sind. Um dies zu demonstrieren wurde der Multiple-Choice-Fragetyp beispielhaft umgesetzt.

Dieser funktionale Durchstich soll die Möglichkeiten und Probleme, die beim Erweitern des Systems auftreten, aufzeigen. Die verschiedenen Anwendungen des Systems (Server, Studenten-Client und Professor-Client) werden getrennt betrachtet.

6.2 Ablauf

Begonnen wurde mit der Implementierung der Server-Anwendung, da diese die Grundlage bildet und die Logik beinhaltet. Es wurde sich für den Anfang darauf beschränkt, grundlegende Funktionen, wie das Anlegen von Daten oder auch die Authentifizierung von Benutzern zu implementieren. Darauf aufbauend wurden im Anschluss die Client- und Professor-Anwendung dahingehend realisiert, dass sie die ersten angebotenen Funktionen nutzen um Verbundstests zu ermöglichen und so einen fachlichen Durchstich zu erreichen. Vorteil an dieser Herangehensweise war die Möglichkeit, früh eventuelle Probleme in der technischen Realisierung zu erkennen und zu beheben.

Nach erfolgreichem Testen dieser Grundlage wurden weitere Anwendungsfälle implementiert.

6.3 Server

Die Server-Anwendung wurde entsprechend der Spezifikation und des Entwurfs realisiert. Im Folgenden werden einzelne wichtige Aspekte der Realisierung näher betrachtet.

6.3.1 Logik zur Verarbeitung eines anonymen Logins

Eine Besonderheit stellt der anonyme Login dar. Dieser soll die Teilnahme von Clients ohne Anmeldung ermöglichen. Für die Statistik und die temporäre Anmeldung an einem Vorlesungstermin muss aber, entsprechend des Entwurfs, ein Benutzer angelegt werden. Realisiert wurde dies, indem ein Client bei einem anonymen Login einen Benutzeraccount erhält, der keinerlei Rückschlüsse auf den Nutzer zulässt. Um diesem Account eine eindeutige Email zuzuordnen, wird eine UUID (Universally Unique Identifier) für den „local-part“ erzeugt. Der „domain-part“ der Email entspricht der URL des Servers. Zusätzlich wird ein zufälliges zehnstelliges Passwort erzeugt. Die beispielhafte Implementierung ist in Abbildung 6.1 dargestellt.

```
1  private BenutzerTyp erzeugeAnonymenBenutzer(RolleTypEnum rolleTyp) {
2      //Zufaelliges Passwort erzeugen
3      String passwort = RandomStringUtils.randomAlphanumeric(10);
4      Login login = new Login(passwort);
5      Rolle rolle = this.rolleVerwalter.findRolle(rolleTyp);
6
7      //Eindeutige UUID erzeugen
8      UUID benutzerId = UUID.randomUUID();
9
10     Benutzer benutzer = new Benutzer(new EmailTyp(benutzerId +
           "@hawclicktracker.com"), login, rolle);
11     benutzerVerwalter.save(benutzer);
12
13     return benutzer.erzeugeTransportObjekt();
14 }
```

Listing 6.1: Beispiel für den Anwendungsfall zum Erstellen eines anonymen Benutzers

Die Entscheidung, wie oft ein neuer anonymer Account erzeugt wird, liegt bei den Clients. In der Beispielimplementierung des Android-Clients in dieser Arbeit wird ein solcher Account einmalig nach der Installation der Anwendung angelegt und bei jedem weiteren anonymen Login weiterverwendet.

6.3.2 Security Constraints zur Einschränkung des Ressourcenzugriffs

Um das Beispiel der anonymen Registrierung aus Punkt 6.3.1 weiter zu vervollständigen, wird im Folgenden die Kommunikation näher betrachtet. Um den Zugriff auf die Ressourcen einzuschränken und diese gezielt Benutzergruppen verfügbar zu machen, werden Security Constraints verwendet. In Listing 6.2 ist der zentrale Security Constraint dieses Systems dargestellt. Mit dem „url-pattern“ /* gilt der Constraint für alle Ressourcen relativ zur Basis

URL der Anwendung. Mit Hilfe von „role-name“ werden alle Rollen festgelegt, denen der Zugriff ermöglicht werden soll. Weitere Einschränkungen sind durch Annotationen bei der Definition der Ressourcen selbst möglich.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Secure</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>ADMINISTRATOR</role-name>
        <role-name>PROFESSOR</role-name>
        <role-name>STUDENT</role-name>
        <role-name>ANONYM</role-name>
    </auth-constraint>
</security-constraint>
```

Listing 6.2: Beispiel eines Security Constraints dieses Systems

6.3.3 Realisierung einer REST-Ressource

In Listing 6.3 ist die Implementierung der Ressource für die anonyme Registrierung gezeigt. Es wurde mit Jersey, der Referenzimplementierung von JAX-RS, gearbeitet, welches Annotationen zur Vereinfachung bereit stellt. „@POST“ legt fest, dass alle POST-Anfragen an diesen durch „@Path“ festgelegten Pfad in dieser Methode bearbeitet werden. Hier sind als Annotationen alle Standardverben von HTTP verfügbar. Mit „@Produces“ wird der Rückgabetyp festgelegt, der in diesem Fall JSON entspricht, da das Benutzer-Transportobjekt zurückgegeben wird.

Schlussendlich besteht noch die Möglichkeit, den Zugriff auf bestimmte, im Security Constraint definierte, Benutzergruppen durch Annotationen wie beispielsweise „@RolesAllowed("PROFESSOR")“ einzuschränken. Der anonyme Login stellt aber insofern einen Sonderfall dar, als es die einzige Ressource des Systems ist, welche ohne Authentifizierung verwendbar sein muss. Hierfür wurde zusätzlich zum in Listing 6.2 dargestellten Security Constraint ein weiterer Constraint erstellt, der eine POST-Anfrage für diese Ressource auch Nutzern ohne Authentifizierung ermöglicht.

In der Implementierung der Ressource selbst wird dann über die Fassade des Anwendungskerns ein anonymer Benutzeraccount erstellt und entsprechend eine Antwort an den anfragenden Client zurückgeschickt. Die Antworten bestehen aus einem HTTP-Statuscode

und gegebenenfalls einem Objekt, dessen Typ, wie zuvor beschrieben, mittels Annotationen festgelegt ist.

An HTTP-Statuscodes kommen in diesem Beispiel Code 200 für die erfolgreiche Verarbeitung und Code 500 für eine nicht-erfolgreiche Verarbeitung zum Tragen. Bei Code 200 wird der Antwort zusätzlich die durch Google Gson erzeugte JSON Darstellung des erzeugten Benutzer-Transportobjekts angehängt.

```
1  @POST
2  @Path("/anonymerbenutzer")
3  @Produces("application/json")
4  @PermitAll
5  public Response erzeugeAnonymenBenutzer() {
6     BenutzerTyp erzeugterAnonymerBenutzer = this.fassade.
          erzeugeAnonymenBenutzer(RolleTypEnum.STUDENT);
7
8      if (erzeugterAnonymerBenutzer != null) {
9          return Response.ok(gson.toJson(erzeugterAnonymerBenutzer),
10             MediaType.APPLICATION_JSON).build();
11     } else {
12         return Response.status(Response.Status.INTERNAL_SERVER_ERROR)
13             .build();
14     }
15 }
```

Listing 6.3: Beispiel einer REST-Ressource für die anonyme Registrierung

6.3.4 Partial Updates: Optimierung des REST-Ressourcen-Designs

PUT ist das HTTP-Standardverb zum Aktualisieren einer Ressource (Tilkov, 2011, S. 54). Hierbei ist aber zu beachten, dass immer das ganze Objekt gesendet werden muss und aktualisiert wird. Dies erzeugt großen Overhead, der in manchen Fällen nicht nötig ist. Ein Beispiel aus dieser Realisierung sei das Starten einer Aufgabe, durch welches der Status der Aufgabe von nicht-aktiv (Attribut istAktiv = false) auf aktiv (Attribut istAktiv = true) geändert wird. Hier wäre es sinnvoll, lediglich den neuen Status zu übertragen und zu aktualisieren.

Das große Problem, welches in der Folge auftrat, war das Fehlen eines HTTP-Verbs für die partielle Aktualisierung von Ressourcen. Der naheliegendste Ansatz, das Weglassen von Attributen in einem PUT, wurde verworfen, da es keine eindeutige Aussage zum Verhalten trifft. Die weggelassenen Argumente könnten für das Objekt auf „null“ gesetzt werden, oder es könnte nur das übertragene Attribut aktualisiert werden. Letzteres ist zwar gewünscht, ist aber aus dem Aufbau der Ressource nicht ersichtlich (Tilkov, 2011, S. 63).

Letztendlich wurde eine Aufteilung der Ressource in Subressourcen umgesetzt. Die Lösung würde beim beschriebenen Beispiel des Startens einer Aufgabe bedeuten, dass ein PUT-Request auf die Subressource „istaktiv“ der Ressource der zu aktualisierenden Aufgabe erfolgt. Basierend auf dem Ressourcen-Design dieser Anwendung würde der Pfad wie folgt aussehen: „/aufgabe/{aufgabeid}/istaktiv“. Die entsprechende Repräsentation der Ressource in WADL Darstellung ist in Listing 6.4 gezeigt.

```
<resource path="aufgabe">
    <resource path="/{aufgabeid}/istaktiv">
        <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="aufgabeid"
            style="template" type="xs:string"/>
        <method id="putAufgabeIstAktiv" name="PUT">
            <request>
                <representation mediaType="application/json"/>
            </request>
            <response>
                <representation mediaType="*/*"/>
            </response>
        </method>
    </resource>
</resource>
```

Listing 6.4: WADL-Darstellung der „istaktiv“-Subressource einer Aufgabenressource

6.4 Studenten-Client

Der Studenten-Client wurde in dieser Arbeit beispielhaft als Android-Applikation entwickelt und wird im Folgenden in Teilen betrachtet.

6.4.1 API Level

Das erste große Problem, welches bei der Entwicklung einer jeden Android-Applikation auftritt, ist die Auswahl des kleinsten benötigten API-Levels, um die Anwendung zu verwenden. Durch die ständige und schnelle Weiterentwicklung der Android-API werden immer mehr Möglichkeiten geschaffen. Dagegen steht die langsame Verbreitung der Updates für die verschiedenen Geräte durch die Hersteller.

Bei der Entwicklung des Clients dieses Systems konnte auf die meisten neuen Möglichkeiten verzichtet werden, weshalb eine breite Abdeckung von Versionen möglich ist. Eine Übersicht über die Verbreitung der verschiedenen Versionen bietet die Abbildung 6.1.

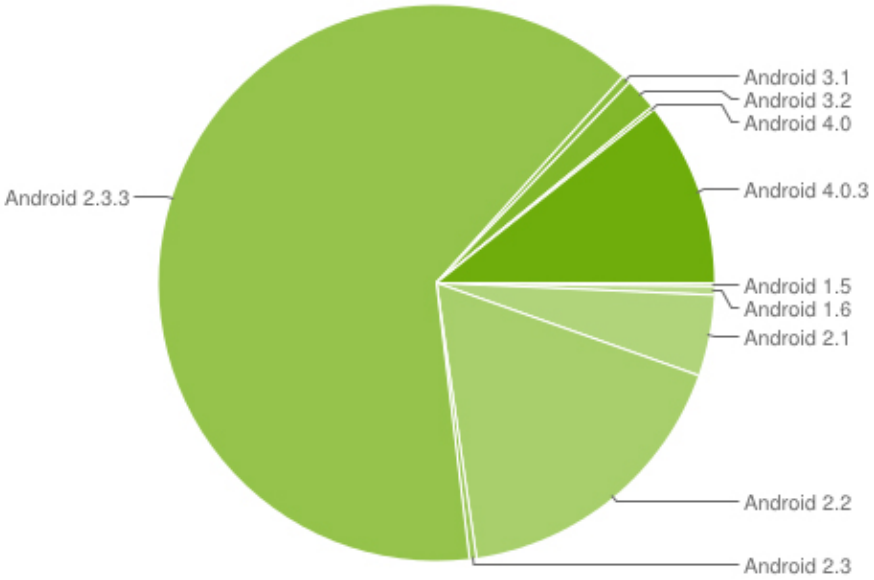

Abbildung 6.1: Platform versions chart (Google Inc., 2012a)

In dieser Applikation wird Version 2.2 vorrausgesetzt, da dies die älteste Version ist, die bereits Google Cloud Messaging for Android unterstützt. Sinn macht dieser Service in dieser Anwendung, da der Server den Android-Client so über neue aktivierte Aufgaben per Push-Service benachrichtigen kann.

Version 2.2 und folgende Versionen entsprechen nach Abbildung 6.2 einer Abdeckung von 94,6% aller derzeit verwendeten Geräte (mit der Einschränkung, dass nur Geräte gezählt wurden, auf denen Google Play verwendet wurde).

Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.2%
1.6	Donut	4	0.5%
2.1	Eclair	7	4.7%
2.2	Froyo	8	17.3%
2.3 - 2.3.2	Gingerbread	9	0.4%
2.3.3 - 2.3.7		10	63.6%
3.1	Honeycomb	12	0.5%
3.2		13	1.9%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.2%
4.0.3 - 4.0.4		15	10.7%

Abbildung 6.2: Platform versions table (Google Inc., 2012a)

6.4.2 Kommunikation mittels AsyncTasks

Die Anwendung selbst ist ein Thin-Client, bei dem die Kommunikation mit dem Server im Vordergrund steht. Um die Benutzeroberfläche während der Kommunikation nicht zu blockieren, wurden sämtliche HTTP-Aufrufe in AsyncTasks gekapselt. Aus den AsyncTasks lässt sich nach Abschluss der Hintergrundoperation die Oberfläche entsprechend der Ergebnisse steuern. Da die REST-Aufrufe für den Ablauf der Anwendung essentiell sind, wird während der Ausführung keine weitere Interaktion mit der Anwendung ermöglicht und ein Lade-Dialog angezeigt.

Als Beispiel sei in Listing 6.5 eine gekürzte Fassung des AsyncTasks für die Registrierung gezeigt. Die RegisterTask Klasse erweitert die generische AsyncTask Klasse, welche drei Typen verwendet. Der erste Typ steht für den Typ der Parameter, welche dem Task zur Ausführung übergeben werden können. Er entspricht in diesem Beispiel dem BenutzerTyp-Transportobjekt des Benutzers, der sich registrieren möchte. Der zweite Typ ist für dieses Beispiel nicht relevant

```
1  private class RegisterTask extends AsyncTask<BenutzerTyp, Void, Boolean> {
2     @Override
3     protected void onPreExecute() {
4        dialog = ProgressDialog.show(CONTEXT, "Registrierung",
5                    "Verbinden...");
6     }
7
8     @Override
9     protected Boolean doInBackground(BenutzerTyp... params) {
10       try {
11          return restService.registriereBenutzer(params[0]);
12       } catch (IOException e) {
13          // Fehler durchreichen
14       }
15    }
16
17    @Override
18    protected void onPostExecute(Boolean result) {
19       dialog.cancel();
20
21       // Behandlung des Ergebnisses der Registrierung
22    }
23 }
```

Listing 6.5: Beispiel eines AsyncTasks für die Registrierung

und wird durch die Verwendung von Void als nicht verwendet markiert. Der dritte Typ steht für den Ergebnistyp des Tasks. Dieser entspricht hier einem Boolean, der dem Erfolg der Registrierung entspricht.

Weiterhin muss die „doInBackground“-Methode implementiert werden, in welcher die Hintergrundoperationen ablaufen. Diese ruft den gekapselten REST-Aufruf für die Registrierung auf und gibt das entsprechende Ergebnis zurück. Für eine übersichtliche Darstellung wurde die Implementierung der Fehlerbehandlung hier nicht berücksichtigt, ist aber im angehängten Quelltext zu finden.

Die Methode „onPreExecute“ wird vor der Ausführung der doInBackground-Methode ausgeführt und öffnet einen ProgressDialog. Dieser wird wiederum nach Ausführung des Tasks in der „onPostExecute“-Methode geschlossen. Hier findet auch die Behandlung des Ergebnisses statt. Bei Fehlschlagen der Registrierung wird eine Fehlermeldung gezeigt. Bei erfolgreicher Registrierung wird der Benutzer eingeloggt und die Login-Daten werden in den Einstellungen der Applikation gespeichert. Dies ist ebenfalls aus Gründen der Übersichtlichkeit in dem

Quellcode-Beispiel nur durch einen Kommentar gezeigt. Die vollständige Implementierung findet sich auch hier im angehängten Quelltext.

6.4.3 Push-Service

Ein weiterer wichtiger Aspekt der Realisierung ist die Benachrichtigung der Studentenclients nach Aktivierung einer Aufgabe, für deren Vorlesungstermin der angemeldete Benutzer registriert ist. Der Service wurde so entworfen, dass eine Erweiterung um weitere Push-Services für andere Endgeräte möglich ist. In dieser Arbeit liegt der Fokus auf einem Android-Client, daher wurde der „Google Cloud Messaging"-Service[5] verwendet, welcher Push-Nachrichten für Android-Geräte ermöglicht.

Ein Android-Client meldet sich mittels eines Registrars beim Google Service für Push-Nachrichten an. In der Folge bekommt dieser eine Registrations-Id, welche er dem Server dieser Anwendung mitteilt. Der Server hält sowohl diese Registrations-Id als auch die Information, dass es sich um einen Android Client handelt. Wird eine Aufgabe aktiviert, wird dem eigenen Push-Service des Servers eine Liste mit Benutzern übergeben, die für den jeweiligen Vorlesungstermin angemeldet sind. Der Push-Service prüft, welche dieser Benutzer ein Android-Gerät besitzen und für den Push-Service registriert sind. Die Liste mit deren gespeicherten Registrations-Ids wird mitsamt der AufgabeId der aktivierten Aufgabe an den Google Service geschickt, welcher die Geräte über die aktive Aufgabe benachrichtigt.

6.4.4 Oberfläche

Im Gegensatz zur Server-Anwendung (6.3) kommt bei der Studentenanwendung mit der Oberfläche ein weiterer wichtiger Punkt zum Tragen. Eine intuitive Oberfläche fördert die Beteiligung an einem Angebot von interaktiven Inhalten mittels dieses Systems. In Abbildung 6.3 ist als Beispiel die zentrale Activity zur Steuerung der Anwendung gezeigt.

[5] http://developer.android.com/guide/google/gcm/index.html

Abbildung 6.3: Oberfläche der zentralen Activity der Studentenanwendung

6.4.5 Statistik-Darstellung

Für die Darstellung der Benutzer-Statistik ist ein Diagramm die naheliegendste Lösung. Speziell für die Benutzerstatistik wurde sich für ein Kreisdiagramm entschieden, welches das Verhältnis von richtigen Antworten gegenüber falschen Antworten zeigt. Um die Möglichkeit zu bieten weitere Diagramme einfach zu integrieren, wurde eine Bibliothek verwendet. Im Folgenden sind einige dieser Bibliotheken kurz vorgestellt und danach in einer Tabelle gegenübergestellt und bewertet.

AChartEngine AChartEngine[6] ist eine aktiv entwickelte Bibliothek für Diagramme, die die meisten gängigen Diagramm-Arten (z.B. Kreisdiagramme oder Balkendiagramme)

[6] http://www.achartengine.org/

anbietet. Sie wird unter der Apache Lizenz 2.0[7] zur Verfügung gestellt. Ein Vorteil ist damit die Möglichkeit, alles den Wünschen entsprechend anpassen zu können.

Chartdroid Die Chartdroid-Bibliothek[8] bietet wie die AChartEngine-Bibliothek die meisten gängigen Diagrammarten an und ist ebenfalls unter der Apache Lizenz 2.0[7] zur Verfügung gestellt.

Der große Unterschied zur AChartEngine besteht in der Verwendung der Bibliothek. Bei diesem Ansatz müssen die Endanwender die Bibliothek für die Diagramme installieren, welche dann mittels eines Intents aus der entwickelten Anwendung heraus aufgerufen wird.

Das Projekt wird zur Zeit (Stand: 28.07.2012) nicht mehr aktiv weiterentwickelt. Der letzte offizielle Release mit der Version 2.0.0 stammt vom November 2010. Die letzte Änderung am Source-Code wurde im November 2011 vorgenommen.

AndroidPlot AndroidPlot[9] wird wie die AChartEngine-Bibliothek sehr aktiv entwickelt. Im Gegensatz zu den beiden bereits vorgestellten Bibliotheken steht bei AndroidPlot der Quellcode nicht zur Verfügung.

Ein weiterer Nachteil ist die fehlende Implementierung einiger Diagrammarten. Unter anderem ist das sehr verbreitete Kreisdiagramm noch nicht realisiert.

Google Chart Tools Als letzte Option sei hier Google Chart Tools[10] vorgestellt. Dies ist im Gegensatz zu den anderen vorgestellten Ansätzen keine Java-Bibliothek, die eingebunden und verwendet werden kann. Google Chart Tools ist viel mehr ein Ansatz von Google, Daten für Webseiten durch Diagramme aufbereitbar zu machen und diese mittels HTML5 und SVG darzustellen.

Es wird darüber hinaus eine API bereitgestellt, um dynamisch Diagramme mittels eines GET- oder POST-Requests zu erstellen. Dabei werden die Daten und Parameter in die bereitgestellte URL eingebettet. Das so durch Google erstellte Diagramm lässt sich in Android einbetten.

Vorteil dieses Ansatzes wäre die plattformübergreifende Portabilität dieses so erstellten Diagramm-Bildes. Der große Nachteil ist die Abhängigkeit von der durch Google angebotenen API. Bei größeren Änderungen könnte die Statistik-Anzeige der in dieser

[7] http://www.apache.org/licenses/LICENSE-2.0
[8] http://code.google.com/p/achartengine/
[9] http://androidplot.com
[10] https://developers.google.com/chart/

Arbeit entwickelten Anwendung nicht mehr funktionieren. Seit April 2012 gilt diese Art der Erstellung von Google-Charts als obsolet.

Im Folgenden sind die beschriebenen Möglichkeiten nach den Kriterien Optik (1), aktive Entwicklung (2), Lizenz (3) und Dokumentation (4) bewertet („+“ steht für gut, „0“ für neutral und „-“ für schlecht).

Name	1	2	3	4
AChartEngine	0	+	+	0
Chartdroid	+	-	+	+
AndroidPlot	0	+	-	+
Google Chart Tools	+	-	-	+

Tabelle 6.1: Bewertung von Bibliotheken zur Darstellung von Diagrammen unter Android

Basierend auf der grundlegenden Beschreibung und auf der Bewertung in Tabelle 6.1 wurde die AChartEngine-Bibliothek ausgewählt. Diese vereint die meisten Vorteile. Der Nachteil der fehlenden Dokumentation wird durch die Community und vielen Hilfen im WWW ausgeglichen. Wie die Statistik für einen Benutzer unter der Verwendung eines Kreisdiagramms aussieht, ist in Abbildung 6.4 gezeigt.

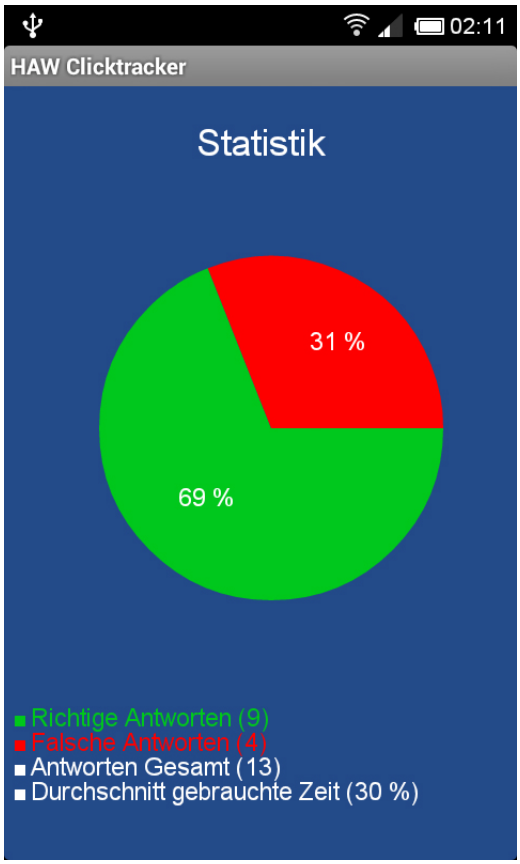

Abbildung 6.4: Oberfläche der Benutzerstatistik-Activity

6.5 Professor-Client

Der Professor-Client wurde in dieser Arbeit beispielhaft als Java SE Anwendung entwickelt. Die Architektur des System ermöglicht, wie auch beim Studenten-Client (6.4), die Entwicklung von Clients für verschiedene Plattformen, die über HTTP kommunizieren können. Auch dieser Client ist wie der Studenten-Client ein Thin-Client. Die Kommunikation mit dem Server steht im Vordergrund und muss ebenfalls nebenläufig zum Thread der Oberfläche laufen, um diese nicht zu blockieren.

Umgesetzt wurde diese Anforderung mit Hilfe eines SwingWorkers, der dem in der Android-Applikation verwendeten AsyncTask in der Verwendung ähnelt. In Listing 6.6 ist die Umsetzung anhand der Authentifizierung in gekürzter Fassung gezeigt. Es wurden zwei zentrale Methoden verwendet. Zum einen die Methode „Boolean doInBackground()“, welche die Aufga-

ben, die im Hintergrund ablaufen sollen, kapselt - in diesem Beispiel ist es ein REST-Aufruf zur Authentifizierung. Zum anderen wird die Methode „void done()“ verwendet, welche mittels des Methodenaufrufs „get()“ das Ergebnis der Hintergrundoperation nach dessen Abschluss erhält und so die Oberfläche entsprechend steuern kann.

```
1  private class CheckLoginWorker extends SwingWorker<Boolean, String> {
2      //Konstruktor zur Übergabe des Login-Objektes
3
4      @Override
5      public Boolean doInBackground() {
6          return restFassade.checkLogin(login);
7      }
8
9      @Override
10     public void done() {
11         try {
12             boolean loginErfolgreich = get();
13
14             //Verarbeitung des Ergebnisses
15         } catch (InterruptedException | ExecutionException ex) {
16             //Fehlerbehandlung
17         }
18     }
19 }
```

Listing 6.6: Beispiel eines SwingWorkers für die Authentifizierung

7 Tests

Es wurden sowohl die verschiedenen Anwendungen des Systems für sich getestet als auch das System im Verbundtest. Besonderer Wert lag hierbei auf den Tests der Server-Anwendung, da diese die Applikationslogik beinhaltet.

7.1 Server

Die Funktionalität wird in der Server-Anwendung in erster Linie durch Komponententests sichergestellt. Hierfür wurde ein Mocking-Framework verwendet (siehe 5.3.4), um eine saubere Trennung der Komponenten beim Testen zu erreichen. Modultests gibt es nicht, da die einzelnen Klassen der Komponenten klein gehalten sind. Gleiches gilt für Integrationstests, da direkt alle vier Komponenten „BenutzerKomponente“, „TerminverwaltungKomponente“, „AufgabeKomponente“ und „StatistikKomponente“ zusammen getestet werden (Systemtest).

Die Komponententests wurden als Black-Box-Tests geschrieben. Hierbei werden die Testfälle aus der Schnittstellenbeschreibung hergeleitet. Die Implementierung der Komponente selbst wird nicht betrachtet. Es wird ausschließlich das Ergebnis, welches die Schnittstelle liefert, betrachtet.

Um die Erstellung der fachlichen Ids zu testen, musste von dem Schema der Black-Box-Tests abgewichen werden, da hierfür Wissen über den inneren Aufbau der Komponente vonnöten ist. Dies fällt in die Kategorie der White-Box-Tests, bei welchen die Kenntnis der Implementierung für die Erzeugung der Tests genutzt wird.

Für die Umsetzung der Tests wurde auf das JUnit-Framework zurückgegriffen. In Listing 7.1 ist ein Test für die Funktionalität zum Erzeugen einer Vorlesung gezeigt.

```
1 @Test
2 public void testErzeugeVorlesung() {
3     String name = "Testvorlesung";
4     VorlesungTyp erzeugteVorlesung = terminverwaltungManager.
          erzeugeVorlesung(name, testbenutzer1.getEmail().getEmail());
5
6     assertNotNull(erzeugteVorlesung);
7     assertTrue(erzeugteVorlesung.getName().equals(name));
8     assertTrue(erzeugteVorlesung.getErstellerEmail().equals(testbenutzer1.
          getEmail().getEmail()));
9 }
```

Listing 7.1: Beispiel eines JUnit-Tests für die Erzeugung einer Vorlesung

Der verwendete Testbenutzer „testbenutzer1“ ist ein Mock-Objekt eines Benutzers. Für Mocks kommt „mockito“ zum Einsatz, welches die intuitive Erstellung von Mock-Objekten ermöglicht.

In Listing 7.2 ist die Erstellung des Benutzer-Mock-Objektes gezeigt. Da die Methode zur Erzeugung einer Vorlesung lediglich die eindeutige Email des Benutzer benötigt, wird ausschließlich diese Methode abgebildet. Und da die Email eines Benutzers ein eigener Typ ist, muss für diesen ebenfalls ein Mock-Objekt erzeugt werden. Sobald die getEmail-Methode des Benutzers aufgerufen wird, gibt das Benutzer-Mock-Objekt das Email-Mock-Objekt zurück. Wird von diesem dann die Emailadresse mittels der getEmail-Methode abgerufen, wird die Emailadresse der Methodendefinition entsprechend als String zurückgegeben.

```
1 Benutzer testbenutzer1 = mock(Benutzer.class);
2 EmailTyp email = mock(EmailTyp.class);
3 when(testbenutzer1.getEmail()).thenReturn(email);
4 when(email.getEmail()).thenReturn("test@name.com");
```

Listing 7.2: Beispiel eines Mock-Objektes für die Benutzer-Entität

7.2 Studenten-Client

Es gibt verschiedene Testverfahren um Android-Anwendungen zu testen. Man kann diese grob in vier Klassen unterteilen (Becker und Pant, 2010, S. 374):

Programmierte Oberflächentests Bei dieser Art des Testverfahrens starten die JUnit-Testfälle die zu testende Oberfläche und steuern diese. Inhalte der Oberfläche können geprüft

und somit die Funktionalität der Oberfläche getestet werden. Es können sowohl einzelne Funktionen der Oberfläche, als auch die gesamte Anwendung gesteuert und getestet werden. Das Testen der ganzen Anwendung wird „smoke testing“ genannt.

Da bei diesen Tests kein Wissen über die Realisierung der Funktionen bekannt sein muss, kann man diese Tests auch der Kategorie der Black-Box-Tests zuordnen.

Modultests für Nicht-Android-Klassen Modultests für Nicht-Android-Klassen setzen voraus, dass es keine Abhängigkeiten zu Funktionen der Android-API gibt. Ist dies der Fall, kann man diese Klassen mit JUnit-Testfällen testen.

Modultests für Android-Komponenten Um Modultests für Android-Komponenten durchzuführen, muss die Systemumgebung simuliert werden, da keine vollständige Laufzeitumgebung gestartet wird.

Stresstests Stresstests setzen die Anwendung über einen bestimmten Zeitraum unter große Last und werten dies aus.

Da der Android-Client aus der Anwendung dieses Systems nur einen Thin-Client darstellt und außer Activities nur einen REST-Konnektor für die Verbindung zum Server vorhanden ist, wurde sich auf programmierte Oberflächentests konzentriert. Der REST-Konnektor wird mit Hilfe von mockito gemockt, was es ermöglicht, die verschiedenen positiven und negativen Fälle der Aufrufe der Activities zu simulieren und entsprechend mit Tests zu prüfen.

Für die Steuerung der Oberfläche wurde das Test-Framework Robotium verwendet. Ein beispielhafter Test ist in Listing 7.3 dargestellt.

„mSimulator“ ist die Steuerung der Activity, welche in diesem Beispiel die Felder für die Emailadresse und das Passwort ausfüllt und anschließend den Button zum Einloggen anklickt. Der Mock des REST-Konnektors gibt für die Prüfung des Logins in jedem Fall „true“ zurück, womit die MainActivity gestartet werden muss, was in diesem Test geprüft wird.

```
@MediumTest
public void testRightLogin() throws Exception {
      when(restManagerMock.checkLogin()).thenReturn(true);

      mSimulator.enterText(0, "testemail");
      mSimulator.enterText(1, "testpasswort");
      mSimulator.clickOnButton("Einloggen");

      assertTrue("MainActivity nicht angezeigt", mSimulator.searchText("
          Abmelden"));
}
```

Listing 7.3: Beispiel eines JUnit-Tests für einen Android-Oberflächentest einer Activity

7.3 Professor-Client

Im Professor-Client werden Verbundstest mit dem Server durchgeführt. Die Anfragen gehen alle vom Client aus, und die Antworten des Servers können entsprechend validiert werden. Zum Einsatz kommt wieder JUnit. Diese Tests sind Black-Box-Tests, die die Annahme treffen, dass die Server-Anwendung die angebotenen Ressourcen implementiert und getestet hat.

8 Zusammenfassung und Ausblick

In diesem Kapitel werden abschließend die erreichten Ergebnisse zusammengefasst, sowie ein Ausblick über mögliche Weiterentwicklungen und Verbesserungen des Systems der Arbeit gegeben.

8.1 Zusammenfassung

In dieser Arbeit wurde ein Clicktracker-System entwickelt, welches interaktive Vorlesungen unterstützen soll.

Um dies zu erreichen, wurde zuerst geklärt, was Clicktracker-Systeme sind, und warum diese in Vorlesungen eingesetzt werden. Weiter wurden bereits vorhandene Lösungen analysiert und zwei potenzielle Nutzer interviewt, um Anforderungen zu finden und das System abzugrenzen.

Basierend auf der Analyse wurde das System durch ein fachliches Datenmodel, Anwendungsfälle und Dialoge spezifiziert. Hierbei wurde Wert auf einen möglichst generischen Aufgabentyp gelegt, damit das System einfach um verschiedene Aufgabentypen erweiterbar ist.

Im nächsten Schritt wurde das System entworfen. Es wurden Komponenten gesucht, sowie der Aufbau und die Konfiguration zur Verbindung beschrieben. Bei der Wahl des Webservices wurde sich für REST entschieden und die Ressourcen und deren Methoden grundlegend festgelegt. Passend dazu wurde JSON als Datenaustauschformat ausgewählt und ein entsprechendes Protokoll entworfen.

Die Architektur entspricht einer 3-Schichten-Architektur, wobei die Schichten teilweise auf verschiedene Knoten verteilt sind. Die Studenten- und Professoranwendungen befinden sich in der Applikationsschicht und sind eigene Knoten. Die Applikationslogik und die Persistenz befinden sich auf dem Server-Knoten und sind durch eine Server-Anwendung realisiert. Gesichert wird die verteilte Anwendung durch die Verwendung von Basic Auth in Verbindung mit HTTPS.

Entsprechend der Spezifizierung und des Entwurfs wurde das System entworfen. Es wurde hierbei mit der Server-Anwendung begonnen, da diese sämtliche Logik kapselt. Nachdem hier

ein Durchstich erreicht wurde, wurde dieser für das gesamte System vorgenommen. Dafür wurde eine Beispielaufgabentyp „Multiple-Choice“ verwendet, und sämtliche Schritte von der Erstellung, über das Ausführen bis zum Auswerten der Aufgabe implementiert. Die Professor-Anwendung wurde als JAVA-Anwendung implementiert und die Studenten-Anwendung als Android-Anwendung. Beide entsprechen Thin-Clients, die mittels HTTP mit dem Server kommunizieren. Sichergestellt wurde die Funktionalität durch JUnit-Tests.

8.2 Ausblick

Das entwickelte System stellt eine Basis für viele Erweiterungsmöglichkeiten dar. Es sind einfach weitere Aufgabentypen integrierbar, genauso wie die Möglichkeit besteht viele verschiedene weitere Studenten-Clients zu entwerfen, um möglichst viele Geräte abzudecken.

Die Statistik bietet einige grundlegende Informationen, ist aber ebenfalls leicht um weitere Attribute und weitere Diagrammarten erweiterbar.

Eine weitere Möglichkeit der Verbesserung würde eine eigene Autorisierungs-Komponente darstellen, um eine feingranularere Rechteverwaltung zu ermöglichen.

Abkürzungsverzeichnis

CRS Classroom Response System, Seite 3

DVM Dalvik Virtual Machine, Seite 6

HTTP Hypertext Transfer Protocol, Seite 9

HTTPS Hypertext Transfer Protocol Secure, Seite 45

JVM Java Virtual Machine, Seite 6

REST Representational State Transfer, Seite 9

SMTP Simple Mail Transfer Protocol, Seite 9

URI Uniform Resource Identifier, Seite 10

W3C World Wide Web Consortium, Seite 8

WWW World Wide Web, Seite 9

XML Extensible Markup Language, Seite 9

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Glossar

Idempotenz In der Informatik spricht man von der Idempotenz einer Anfrage, wenn diese mehrfach ausgeführt im immer gleichen Ergebnis resultiert. 10

JavaScript Object Notation JSON ist ein Format, welches den Datenaustausch zwischen Anwendungen ermöglicht. Vorteile sind seine einfache und dadurch schnelle Verarbeitung durch Maschinen und die für Menschen einfach lesbare Darstellung. 10

JavaServer Pages JSP ist eine Technologie zur Erstellung von dynamischen Webinhalten. 11

Universally Unique Identifier UUID ist ein durch die Open Software Foundation standartisierter Identifikator, der eine eindeutige Kennzeichnung von Daten ermöglicht. 47

Web Application Description Language WADL ist eine Beschreibungssprache in XML-Syntax, die speziell für REST und HTTP entworfen wurde. Sie soll Ressourcen, deren unterstützte Verben und die Formate zum Austausch der Daten, in maschinenlesbarer Form darstellen (Tilkov, 2011, S. 147). 50

Web Service Description Language WSDL ist eine Metasprache, die die Funktionalität eines Webservices beschreibt. Sie ist plattform- und programmiersprachenunabhängig. 9

Literaturverzeichnis

[Becker und Pant 2010] Becker, Arno ; Pant, Marcus: *Android 2: Grundlagen und Programmierung*. 2. Heidelberg : dpunkt-Verl, 2010. – ISBN 978-3-89864-677-2

[Booth u. a. 2004] Booth, David ; Haas, Hugo ; McCabe, Francis ; Newcomer, Eric ; Champion, Michael ; Ferris, Chris ; Orchard, David: *Web Services Architecture*. 2004. – URL http://www.w3.org/TR/ws-arch/. – Zugriffsdatum: 25.04.2012

[Bruff 2010] Bruff, Derek: *Classroom Response Systems ("Clickers")*. 2010. – URL http://cft.vanderbilt.edu/teaching-guides/technology/clickers/. – Zugriffsdatum: 24.04.2012

[Chickering und Gamson 1987] Chickering, Arthur W. ; Gamson, Zelda F.: *Seven Principles for Good Practice in Undergraduate Education*. 1987. – URL http://www.uis.edu/liberalstudies/students/documents/sevenprinciples.pdf. – Zugriffsdatum: 28.05.2012

[Dobjanschi 2010] Dobjanschi, Virgil: *Developing Android REST client applications*. 2010. – URL http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html. – Zugriffsdatum: 31.03.2012

[Fielding u. a. 1999] Fielding, R. ; Irvine, UC ; Gettys, J. ; Compaq/W3C ; Mogul, J. ; Compaq ; Frystyk, H. ; W3C/MIT ; Masinter, L. ; Xerox ; Leach, P. ; Microsoft ; Berners-Lee, T.: *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1*. 1999. – URL http://tools.ietf.org/html/rfc2616. – Zugriffsdatum: 03.05.2012

[Fielding 2000] Fielding, Roy T.: *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, UNIVERSITY OF CALIFORNIA, IRVINE, Dissertation, 2000. – URL http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. – Zugriffsdatum: 31.03.2012

[Fowler 2004] Fowler, Martin: *Inversion of Control Containers and the Dependency Injection pattern*. 2004. – URL http://www.martinfowler.com/articles/injection.html. – Zugriffsdatum: 06.08.2012

[Goncalves 2010] Goncalves, Antonio: *Beginning Java EE 6 platform with GlassFish 3: From novice to professional*. 2. Norwood Mass : Books24x7.com, 2010. – ISBN 143022889X

[Google Inc. 2012a] Google Inc.: *Platform Versions*. 2012. – URL http://developer.android.com/about/dashboards/index.html. – Zugriffsdatum: 16.07.2012

[Google Inc. 2012b] Google Inc.: *What is Android?* 2012. – URL http://developer.android.com/guide/basics/what-is-android.html. – Zugriffsdatum: 08.05.2012

[Gudgin u. a. 2007] Gudgin, Martin ; Hadley, Marc ; Mendelsohn, Noah ; Moreau, Jean-Jacques ; Nielsen, Henrik F. ; Karmarkar, Anish ; Lafon, Yves: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 2007. – URL http://www.w3.org/TR/soap12-part1/. – Zugriffsdatum: 22.05.2012

[H-ITT 2010] H-ITT: *Student Response Systems*. 2010. – URL http://www.h-itt.com/higher-education/student-response-systems.htm. – Zugriffsdatum: 24.04.2012

[Jendrock u. a. 2012] Jendrock, Eric ; Cervera-Navarro, Ricardo ; Evans, Ian ; Gollapudi, Devika ; Haase, Kim ; Oliveira, William M. ; Srivathsa, Chinmayee: *The Java EE 6 Tutorial*. 2012. – URL http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf. – Zugriffsdatum: 19.07.2012

[Kehle u. a. 2008] Kehle, Markus ; Hien, Robert ; Röder, Daniel: *Hibernate und die Java Persistence API: Einstieg und professioneller Einsatz*. 2. Unterhaching : Entwickler Press, 2008. – ISBN 978-3-86802-014-4

[Nurseitov u. a. 2009] Nurseitov, Nurzhan ; Paulson, Michael ; Reynolds, Randall ; Izurieta, Clemente: *Comparison of JSON and XML Data Interchange Formats: A Case Study*. 2009. – URL http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf. – Zugriffsdatum: 17.07.2012

[Petereit 2005] Petereit, Armin: *Einfluss von Lernumgebungen auf Lehrqualität und Lernmotivation*. München, Ludwig-Maximilians-Universität München, Dissertation, 2005. – URL http://edoc.ub.uni-muenchen.de/3863/. – Zugriffsdatum: 27.05.2012

[Siedersleben 2004] Siedersleben, Johannes: *Moderne Softwarearchitektur: Umsichtig planen, robust bauen mit Quasar*. 1. Heidelberg and Neckar : Dpunkt-Verl., 2004. – ISBN 3898642925

[Tilkov 2011] Tilkov, Stefan: *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien.* 2. Heidelberg and Neckar : dpunkt, 2011. – ISBN 978-3-89864-732-8

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

Hamburg, 23. August 2012 Andre Ziehn