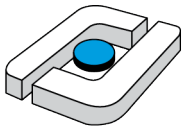

Hochschule Osnabrück
University of Applied Sciences

Fakultät
Ingenieurwissenschaften und Informatik

Projektdokumentation

für das
Masterprojekt
mit dem Thema

Entwicklung eines Android Usability Testing Frameworks

Autor: Albert Hoffmann
albert.hoffmann@hs-osnabrueck.de

Prüfer: Prof. Michaela Ramm, M.A.

Abgabedatum: 06.10.2014

Kurzfassung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

Das ganze auf Englisch.

I Inhaltsverzeichnis

II Abbildungsverzeichnis

III Tabellenverzeichnis

IV Listing-Verzeichnis

V Abkürzungsverzeichnis

AES	Advanced Encryption Standard
API	Application Programming Interface
AS	Application Server
CRUD	Create, read, update and delete
DAO	Data Access Object
DBMS	Database Management System
HATEOAS	Hypermedia as the Engine of Application State
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JAX-RS	Java API for RESTful Web Services
JDBC	Java Database Connectivity
JDK	Java SE Development Kit
JNI	Java Native Interface
JPA	Java Persistence API
JSF	JavaServer Faces
JSON	JavaScript Object Notation
JSP	JavaServer Pages
ORM	Object-relational mapping
POJO	Plain Old Java Object
REST	Representational State Transfer
RPC	Remote Procedure Call
SDK	Software Development Kit
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SQL	Structured Query Language
URL	Uniform Resource Locator

1 Einleitung

1.1 Motivation

Bei begrenztem Budget, z. B. von Studierenden sind umfangreiche Labortests oder das Beauftragen von Dienstleistern oft nicht möglich.

Ein weiterer Faktor ist auch die Zielgruppe für die entwickelte Anwendung. Bei räumlich verteilten Anwendern sind Labortests durch den logistischen Aufwand nur schwer möglich und auch die finanzielle Belastung steigt durch Transportkosten und evtl. mobiles Equipment.

Deshalb wurde im Wintersemester 2013/2014, zusammen mit Oliver Erxleben, ein Projekt durchgeführt, mit dem Ziel Usability-Tests mit minimalem Aufwand für den Teilnehmer durchzuführen (siehe Abschnitt 1.4).

Die damals gesammelten Erkenntnisse und Erfahrungen sollen nun vertieft werden, mit dem Ziel die bestehenden Prototypen zu einem nutzbaren Framework weiter zu entwickeln.

1.2 Zielsetzung

Das Ziel ist es, ein Usability-Testing Framework oder Toolkit zu entwickeln, mit dem auch einzelne Entwickler oder kleine Teams ihre Anwendungen und Prototypen mit der Hilfe von Testbenutzern testen können. Für den eigentlichen Testablauf beim Benutzers soll nur ein Android basiertes Gerät und eine Internetverbindung benötigt werden. Ziel dabei ist es, das Testen möglichst einfach zu gestalten, ohne Labor und zusätzliches Equipment.

Auf der Seite des Testleiters soll nur ein PC oder Notebook benötigt werden, ohne zusätzliche Hardware.

1.3 Problemstellung

1.4 Stand bei Projektbeginn

Dieses Projekt basiert auf einer Arbeit, welche zusammen mit Oliver Erxleben (oliver.erxleben@hs-osnabrueck.de) im Modul *Mensch-Maschine-Kommunikation* (Wintersemester 2013/2014) des Informatik–Masterstudiengangs *Verteilte und mobile Anwendungen* erstellt wurde. Die Ergebnisse zum Stand der Abgabe können aus dem Git-Repository von Oliver Erxleben unter der folgenden URL[1] abgerufen werden:

`https://github.com/olivererxleben/hipsterbility/releases/tag/v1.0_semester_final`

Der Vollständigkeit halber wird der Stand der vorausgehenden Arbeit an dieser Stelle kurz wiedergegeben. Es wurde ein Prototyp erstellt, welcher aus einer Android Bibliothek und einem kleinen REST[2]-Server besteht. Zum Testen der Bibliothek wurde weiterhin eine kleine Android Applikation erstellt. Insgesamt wurde folgender Funktionsumfang skizziert:

- Android 4.x Bibliothek

[1] Uniform Resource Locator
[2] Representational State Transfer

  - Abrufen von Testsitzungen und Testaufgaben vom Server.
  - Benutzeroberfläche zum Auswählen und Starten von Testsitzungen.
  - Aufzeichnen des Kamerabildes der Frontkamera mit Ton, nach dem Starten einer Testsitzung.
  - Erstellen von Screenshots der Applikation bei Benutzereingaben auf dem Touchscreen und Visualisierung der Eingaben auf dem Screenshot.
  - Hochladen der gesammelten Daten zum Server.
- Node.js basierter Server mit REST-Schnittstelle und Webinterface
  - Bereitstellen von Testsitzungen und -aufgaben.
  - Rudimentäres Benutzermanagement.
  - Weboberfläche zum Erstellen von Testsitzungen und -aufgaben
  - Speichern der hochgeladenen Daten in einer Datenbank und im Dateisystem.
  - Aufbereiten der Screenshots und des Videos von der Frontkamera in ein Ergebnisvideo für die Auswertung.

Die Idee und die ursprüngliche Planung stammen von Oliver Erxleben, welcher auch für die Serverkomponente zuständig war.

Da viele Funktionen in der Implementierung nur skizziert wurden eignet sich der Prototyp nicht für die Verwendung in einer produktiven Umgebung.

Folgende Komponenten wurden aus der vorhergehenden Arbeit übernommen und weiterentwickelt:

- das grundlegende Konzept,
- Teile des Datenmodells,
- und Quellen der Android Bibliothek und Testapplikation.

Der Server wird auf der Basis von *Java EE 7* neu aufgebaut.

2 Stand der Technik

2.1 Mobile Usability Testing Methoden

Usability-Tests können grob in zwei räumliche Kategorien eingeteilt werden, Labortests und Feldtests, welche jeweils moderiert oder unmoderiert stattfinden können.

Ein einfacher Testaufbau für Labortests lässt sich z.B. mit Hilfe eines Computers mit angeschlossener, externer Kamera realisiert werden. Vorausgesetzt werden auch entsprechende Räumlichkeiten und Benutzer, welche die Tests durchführen. Die, an den Computer angeschlossene Kamera, dient zum Aufzeichnen und Dokumentieren des Bildschirms eines mobilen Gerätes auf dem getestet wird. Über die Kamera werden auch die Benutzereingaben dokumentiert. [Vgl. Bud14]

2.1.1 Thinking Aloud

2.2 Mobile Usability Testing Anbieter

Neben Methoden und Werkzeugen zum Messen und Testen von Software-Usability gibt es auch Dienstleister, deren Geschäftsmodell auf Usability-Tests und Untersuchungen basiert. Hier werden einige Anbieter solcher Dienstleister kurz vorgestellt und das Angebot kurz beschrieben. Die Auswahl und Beschreibung beschränkt sich auf Anbieter, die das Testen von mobile Applikationen anbieten.

2.2.1 UserTesting

Der Dienstleister *UserTesting* (`http://www.usertesting.com/`) bietet, unter anderem, Usability-Studien an, welche von Testteilnehmern durchlaufen werden. Das Angebot beinhaltet auch das Testen von mobilen Webseiten und Applikationen. [Vgl. Use14c]

Der Kunden/Auftraggeber stellt Testaufgaben zusammen, welche die Testpersonen erfüllen sollen. Es gibt die Möglichkeit Geräteklasse, Anzahl der Testteilnehmer und eine Zielgruppe festzulegen. Das Ergebnis der in Auftrag gegebenen Studie kann, laut Anbieter, schon nach ca. einer Stunde vorliegen und umfasst Videos von Testteilnehmern währen der Tests, schriftliche Antworten und die Möglichkeit der anschließenden Befragung der Teilnehmer. [Vgl. Use14a]

Das Angebot zum Testen von mobilen Applikationen umfasst die Plattformen Android und iOS. Pro Plattform werden als Testgeräte Tablets und Smartphones angeboten, welche bei der Spezifikation einer Teststudie angegeben werden. Auch hier werden Zielgruppen ausgewählt und Testaufgaben spezifiziert. Das Testergebnis umfasst auch eine Videoaufzeichnung vom Test, auf dem der Gerätebildschirm sichtbar ist und der Benutzer Aussagen nach der *Thinking Aloud* Methode (siehe Abschnitt 2.1.1) tätigt. Die pauschalen Kosten für kleine Studien belaufen sich auf $49 pro Testbenutzer. [Vgl. Use14b]

2.2.2 UserZoom

Das *UserZoom* (`http://www.userzoom.de`) Angebot vereint Werkzeuge und Dienstleistungen. Letztere sind vorwiegend unterstützender Natur und umfassen z.B. Beratung, Rekrutierung von Teilnehmern und den Kunden-Support [vgl. Use13a].

Das Angebot bzgl. des Mobile Testing umfasst eine mobile Applikation für die iOS und Android Plattformen, welche sich zum Testen von Webseiten und webbasierten App-Prototypen eignet. Das Verfahren wird als „[...] Remote Unmoderated Mobile Usability Testing [...]“ [Use13b] bezeichnet und erlaubt ein standortunabhängiges Testen. Angemeldete Testpersonen werden in Zielgruppen eingeteilt und per E-Mail zu Studien und Befragungen eingeladen. Das eigentliche Testen wird über eine native mobile Applikation durchgeführt, in welcher Fragen beantwortet und Aufgaben erfüllt werden sollen. Mit der Applikation lassen sich keine nativen Applikationen testen. [Vgl. Use13b]

Im Gegensatz zu *UserTesting* gibt *UserZoom* keine Pauschalpreise an. Ein Rechenbeispiel auf der Webseite des Unternehmens vergleicht Remote Usability Testing mit Labortests. Dabei

werden Kosten von bis zu 120 € pro Testbenutzer in Labortests und ca. 10 € für Remote-Testbenutzer berechnet. Die Kostengegenüberstellung am Ende des Artikels, welche eine hypothetische Ersparnis von 40 % berechnet lässt mit 12 440 € zu 7600 € darauf schließen, dass sich das Angebot eher an Unternehmen richtet, als an einzelne Entwickler oder kleine Teams. [Vgl. Dar13]

2.3 Mobile Usability Testing Werkzeuge

2.4 Relevante Technologien

2.4.1 Chromecast Screen Mirroring

2.4.2 Android Bildschirmaufzeichnung

3 Konzept

3.1 Anforderungen

An das Projekt ergeben sich die folgenden Anforderungen:

- Einfache Benutzung
- Einfache Integration in bestehende Apps

3.2 Architektur

3.2.1 Übersicht

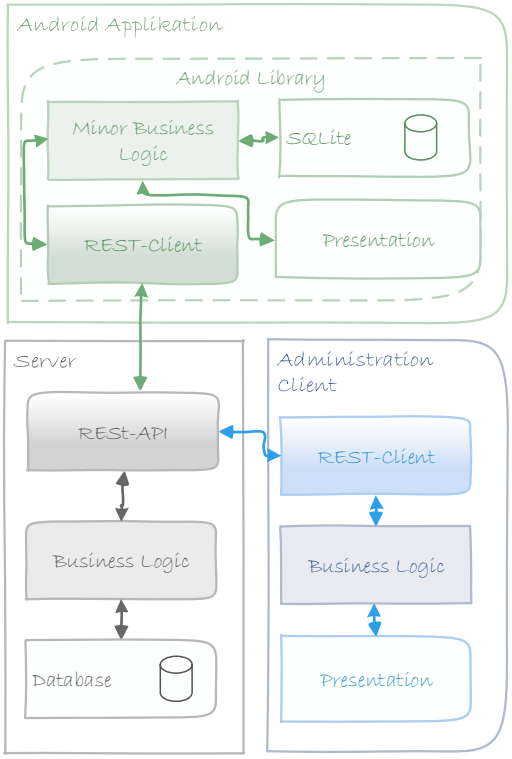

Abbildung 1: Architektur des Frameworks (Entwurf).

Das Framework ist in drei Komponenten unterteilt: Server, Android Bibliothek und ein Client für Testleiter / Administratoren.

3.2.2 Server

Der Aufbau der Komponenten ist in Abbildung 1 dargestellt. Die Kommunikation zwischen den Komponenten erfolgt per HTTP[3] und eine REST-API[4]. Der Server nutzt zur Datenhaltung eine Datenbank, auf welche durch die Geschäftslogik zugegriffen wird.

3.2.3 Android-Biliothek

3.2.4 Administrations-Client

3.3 Evaluierung und Auswahl von Technologien

Im Gegensatz zu dem Ursprünglichen Projekt (siehe Abschnitt 1.4) liegt hier nicht der Fokus

[3] Hypertext Transfer Protocol
[4] Application Programming Interface

auf der möglichst schnellen und einfachen Entwicklung von Prototypen, sondern eines weitgehend einsatzfähigen Frameworks.

Aus diesem Grund werden Technologien unter verschiedenen Gesichtspunkten neu evaluiert und ausgewählt.

Da das Framework später quelloffen und kostenlos bereitgestellt werden soll, wird der Einsatz kommerzieller Software und Bibliotheken von Anfang an ausgeschlossen.

Nachfolgend werden die einzelnen Komponenten des Frameworks näher betrachtet und mögliche Technologien, sowie die finale Auswahl, dargestellt.

Besonderes Gewicht bei der Auswahl bekommt die Zielgruppe, welche aus Android-Entwicklern besteht. Android Applikationen werden vorwiegend in der Java-Programmiersprache entwickelt, weshalb zumindest deren Grundkenntnisse vorausgesetzt werden können. Aus den Android SDK[5] Systemvoraussetzungen [vgl. And14] geht außerdem hervor, dass für das Entwickeln ein Computer vorhanden sein muss, welcher als Betriebssystem ein halbwegs aktuelles Windows, Max OS X oder Linux besitzt, auf dem mindestens das JDK[6] 6 vorhanden ist.

Begonnen wird mit der Auswahl der Server-Plattform, da diese die Auswahl der Technologie für den Testleiter-Client beeinflusst. Besonderer Wert wird auch auf die schnelle und einfache Entwicklung gelegt.

3.3.1 Server

Da das Framework ein verteiltes System mit klassischer Client-Server Architektur ist (siehe Abschnitt 3.2), wird eine Serverkomponente benötigt.

Aus den allgemeinen Anforderungen, welche in Abschnitt 3.1 aufgestellt wurde, werden nun Kriterien zur Auswahl der Server-Plattform definiert.

Anforderungen an den Server:

- Implementierung einer REST-API,
- Anbindung eines Datenbanksystems und
- Unterstützung der Java Programmiersprache für geteilte Klassen.

Für den Prototyp wurde *Node.js* (`http://nodejs.org/`) eingesetzt. *Node.js* ist ein ereignisgesteuertes Framework, für welches vorwiegend in der JavaScript-Sprache [vgl. Joy14] programmiert wird.

[5] Software Development Kit
[6] Java SE Development Kit

Zwar existiert ein Bridge-API um Java Code in JavaScript verwenden zu können . Die Aufrufe erfolgen intern über JNI[7] und bieten somit keinen Ersatz für eine vollständige Java (EE) Umgebung. [Vgl. Fer14]

Das geteilte Verwenden von Java-Klassen in Client und Server wäre demnach nur eingeschränkt möglich.

Eine Alternative, welche die o.g. Anforderungen erfüllt, sind Java EE Application Server. Sie bieten den vollen Java Sprachumfang und erlauben das Teilen von Klassen zwischen verschiedenen Modulen. Die aktuelle Java EE 7 Spezifikation beinhaltet u. a. *Java Persistence 2.1* für den Datenbankzugriff und die *Java API for RESTful Web Services (JAX-RS) 2.0* zum Aufbau eines RESTful Webservice. [Vgl. Ora14]

Als Java EE 7 Implementierung wird der *Glassfish 4.1* (`https://glassfish.java.net/`) AS[8] verwendet, da er die Referenzimplementierung der Plattform darstellt [vgl. Cli13, S. 14]. Außerdem ist er quelloffen (Open Source Edition) und kostenlos verfügbar.

Basierend auf dieser Auswahl erfolgt im nächsten Abschnitt die Technologie- und Plattformauswahl für den Testleiter-Client.

3.3.2 Testleiter Client

Wie auch beim Server soll der Client nicht von kostenpflichtiger Software oder Bibliotheken abhängen. Es bietet sich an, auch beim Client die Java Programmiersprache als Basis zu nehmen. Der Client ist eine Anwendung mit grafischer Oberfläche. Zum Erstellen einer solchen stehen verschiedenen Standardbibliotheken bereit. Einerseits wäre ein Web-Client denkbar, der auf der Servlet-Engine und JSP[9] bzw. JSF[10]/Facelets basiert. Eine Neuerung in Java EE 7 ist die HTML[11] 5 Unterstützung [vgl. Cli13, S. 5], mit welcher auch die WebSocket Technologie eingeführt wurde. Sie erlaubt eine bidirektionale, ereignisgesteuerte Kommunikation mit niedriger Latenz zwischen Client und Server.

Wie im Konzept beschrieben ist es nötig zwei oder mehr dynamische Inhalte in einer Benutzeroberfläche anzuzeigen und zeitlich zu synchronisieren. Es wurde keine einfache Möglichkeit gefunden dies mit den zuvor genannten Technologien zu erreichen. Es gibt zwar mehrere HTML5 Beispielimplementierungen, die das synchrone Abspielen von zwei oder mehr Videos zeigen sollen [Wal11; Sha11], dabei wurde jedoch jeweils das selbe Video mit der gleichen Bildwiederholrate, Auflösung, Dauer usw. gewählt. Diese Demonstration ist nicht ausreichend um die Entwicklung des Clients zu bestimmen.

Aus diesem Grund gibt es JavaScript Frameworks wie *Popcorn.js* (`http://popcornjs.org/`), die es erlauben z. B. Videos und andere Medien/Inhalte synchron darzustellen. Diese Frame-

[7] Java Native Interface
[8] Application Server
[9] JavaServer Pages
[10] JavaServer Faces
[11] Hypertext Markup Language

work bietet zwar die nötige Funktionalität, erfordert jedoch zusätzlich den Einsatz der JavaScript Sprache und in Kombination mit WebSockets um das gewünschte Ergebnis zu erreichen. [Vgl. bP14]

4 Realisierung des Servers

4.1 Datenbank und Persistenz

4.1.1 Konfiguration des Glassfish AS zur Verwendung einer MySQL Datenbank

4.2 Datenmodell und Datenbankschema

Die Datenbank wurde nach dem „Code first“ Ansatz entwickelt. Diese Vorgehensweise wurde gewählt, da das bisherige Datenmodell erweitert wurde und die Datenbank nach belieben angepasst werden kann, da keine anderen Anwendungen direkt auf die Datenbank zugreifen.

Das Datenmodell besteht aus JPA[12]-Entitätsklassen (Entity). Dies sind mit JPA-Annotationen versehene POJO[13]-Klassen sind [vgl. KS13, S. 17-19].

Das resultierende Datenbankschema (siehe Abbildung 2) wird durch den ORM[14]-Provider erstellt und bedarf keine manuellen Anpassungen, da Schlüssel, Beziehungen und Einschränkungen der Tabellen direkt in den Entitätsklassen mit Annotationen festgelegt werden können.

Lediglich eine View wurde manuell erstellt, um die Vorgabe des Authentifizierungsproviders zu erfüllen (siehe Abschnitt 4.4.2)

[12] Java Persistence API
[13] Plain Old Java Object
[14] Object-relational mapping

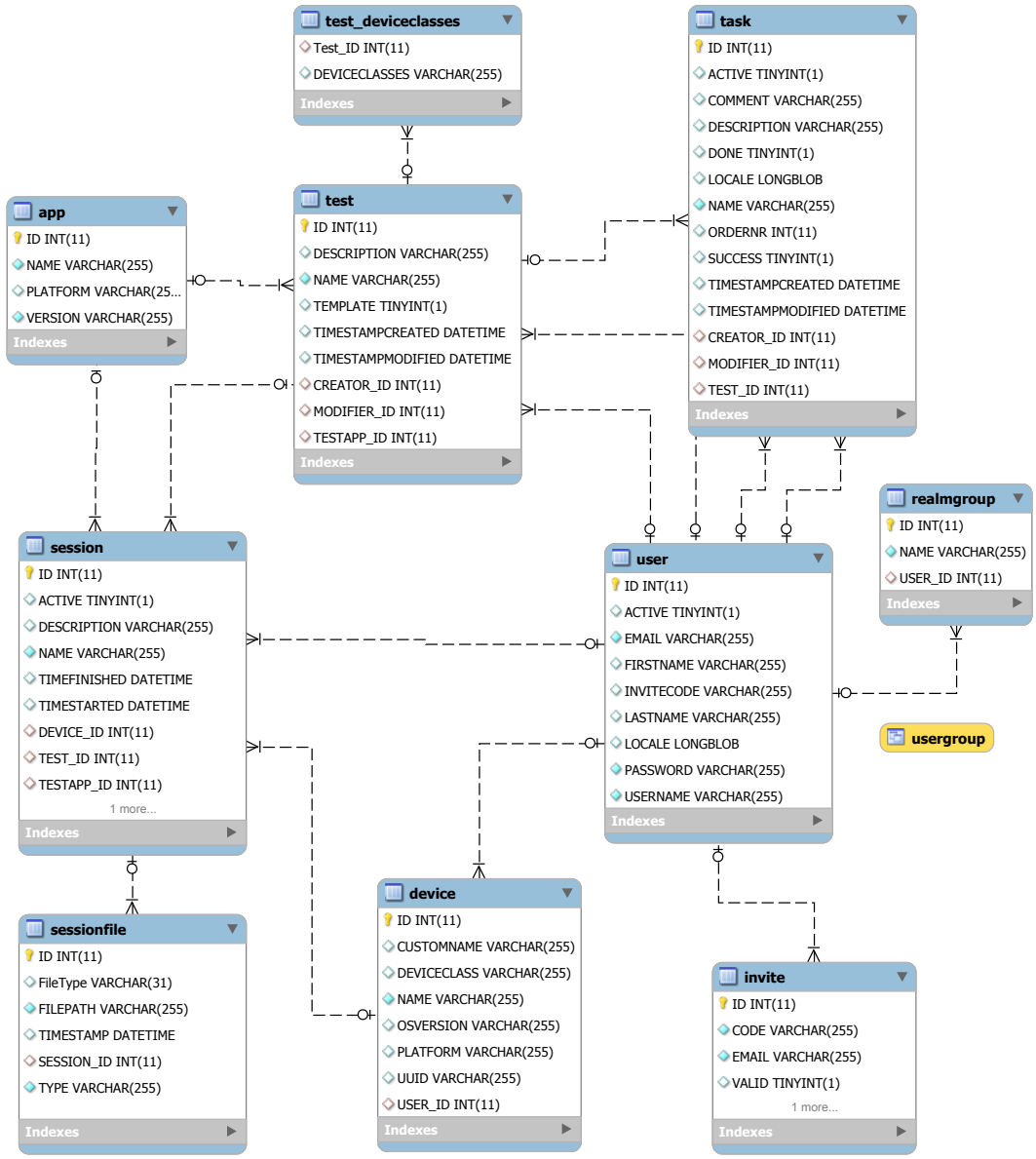

Abbildung 2: Datenbankschema des Hipsterbility-Servers.

Datenbanktabelle	Entitätsklasse(n)	Beschreibung
app	TestAppEntity	Eine im System registrierte Applikation die für Test vorgesehen ist
device	DeviceEntity	Registrierte Geräte der Benutzer
user	UserEntity	Benutzerdaten und Sammlungen von Objekten mit Benutzerbezug
realmgroup	GroupEntity	Rollenzuweisung der Benuter
invite	InviteEntity	Einladung für die Registrierung neuer Benutzer
session	TestSessionEntity	Repräsentation eines durchgeführten Testdurchlaufs
test	TestEntity	Objekt zur Darstellung eines Usability-Tests
task	TaskEntity	Eine Aufgabe in einem Usability-Test, welche vom Testbenutzer ausgeführt werden soll.
test_deviceclasses	*DeviceClass* String enum	Eine Liste mit Geräteklassen für die der jeweilige Test durchgeführt werden soll.
sessionfile	FileEntity Subklassen	Tabelle mit Pfaden und Metainformationen zu Dateien, die vom Android-Client während einer Testsitzung erstellt werden

Tabelle 1: Tabellen und Entitäten der Persiszenzschicht.

4.3 RESTFul Webservice

Um die Android Bibliothek und den Testleiter-Client anzubinden wird eine REST-API verwendet. Da der *Glassfish* AS die Grundlage für die Serverkomponente darstellt, kommt die JAX-RS[15] 2.0 Implementierung *Jersey* (https://jersey.java.net/) zum Einsatz, welche beim AS mitgeliefert wird. Als Referenz für die Implementierung der Schnittstelle und den Aufbau der Ressourcen dient *RESTful Java with JAX-RS 2.0* von Burke [Bur14].

Eine möglichst lose Kopplung der verschiedenen Klassen wird durch die Verwendung von Interfaces und Dependency Injection erreicht. Ein gutes Beispiel dafür zeigt Robert Leggett mit seinem GitHub Projekt *jersey_restful_webservice*[16], welches vom Ressourcenaufbau eher dem REST-RPC[17] Schema folgt.

Aktuell wird nur das JSON[18] Datenformat unterstützt. Für das (Un–)Marshalling wird serverseitig die *Jackson* (https://github.com/FasterXML/jackson) Bibliothek verwendet.

Eine Implementierung des HATEOAS[19] Prinzips [vgl. Bur14, S. 11-13] wurde nicht vorgenommen, da die entwickelte API nicht öffentlich zugänglich sein wird. Außerdem wird sie aktuell nur von den eigenen Clients genutzt, welchen der Aufbau der API nd Ressourcen bekannt ist.

[15] Java API for RESTful Web Services
[16] GitHub Repository: https://github.com/Robert-Leggett/jersey_restful_webservice
[17] Remote Procedure Call
[18] JavaScript Object Notation
[19] Hypermedia as the Engine of Application State

Ein detaillierte Beschreibung der Ressourcen mit Pfaden und HTTP-Methoden befindet sich in Abschnitt 4.3.2.

4.3.1 Java REST-API mit JAX-RS Annotationen und CDI

4.3.2 Implementierte Ressourcen

Die Ressourcen werden als Gruppen von Objekten angesehen und es werden Nomen im Plural verwendet [vgl. Bur14, S. 20 ff.] um die Benennung einheitlich zu gestalten (z.B. `users`, `sessions`, `devices`). Dies weicht vom Datenbankmodell ab, da dort die Tabellen im Singular benannt sind (siehe Abschnitt 4.1).

Wie auch bei den DAO's[20] in der Persistenzschicht der Serveranwendung wurden die vier grundlegenden Datenoperationen CRUD[21] implementiert. Die Zuordnung zwischen Operation und HTTP-Methoden bzw. SQL[22]-Befehlen ist in Tabelle 2 zu sehen.

Operation	**HTTP**	**SQL**
Create	`POST`	`INSERT`
Read / Retrieve	`GET`	`SELECT`
Update / Modify	`PUT`	`UPDATE`
Delete	`DELETE`	`DELETE`

Tabelle 2: Zuweisung der CRUD Operationen und HTTP-Methoden und SQL-Befehlen.

4.3.3 Aufbau von Ressourcen und Zugriffsrechte

Die Ressourcen sind flach gehalten. Je nach Benutzerrolle führen Aufrufe der Methoden zu verschiedenen Ergebnissen. Bei Ressourcen mit Benutzerbezug (z. B. Geräte und Testsitzungen) bekommt ein angemeldeter Benutzer in der Rolle `USER` nur Objekte, welche mit ihm in Relation stehen. Dadurch wird sichergestellt, dass einfache Benutzer nicht auf Daten anderer Benutzer zugreifen können. Benutzer in der Rolle `ADMIN` können hingegen auf alle Objekte in einer Ressource zugreifen.

Das erstellen von Objekten mit Benutzerbezug über die `POST`-Methode erfordert in der Rolle `ADMIN` zusätzlich zu dem zu erstellenden Objekt auch noch die `ID` des Benutzers, für welchen diese erstellt werden soll, sofern die Objekte keine bidirektionale Beziehung haben und sich der Bezug daraus herstellen lässt.

Nachfolgend werden die Resourcen mit den implementierten HTTP-Methoden aufgelistet. Die Pfadangabe erfolgt relativ zur Basis-URL. Angaben in geschweiften Klammern sind Pfad-Parameter, welchen im Aufruf ein Wert zugewiesen ist.

[20] Data Access Objects
[21] Create, read, update and delete
[22] Structured Query Language

Rollen	Methode	Pfad	Beschreibung
USER	GET	/users	Daten des angemeldeten Benutzers
ADMIN	GET	/users	Liste aller Benutzer
USER	PUT	/users	Bestätigung oder Fehlermeldung
ADMIN	PUT	/users/{id}	Bestätigung oder Fehlermeldung
ALL, ADMIN	POST	/users	ID des neu erstellten Benutzers
ADMIN	GET	/users/{id}	Ausgabe des Benutzers mit der ID id
ADMIN	DELETE	/users/{id}	Löschen des Benutzers mit der ID id
USER	GET	/devices	Liste mit Geräten des Benutzers
ADMIN	GET	/devices	Liste mit allen Geräten
ADMIN	GET	/devices/{id}	Gerät mit der ID id
ADMIN	DELETE	/devices/{id}	Gerät mit der ID id
ADMIN	PUT	/devices/{id}	Gerät mit der ID id

Tabelle 3: REST-Ressourcen mit HTTP-Methoden, Pfad, Benutzerrolle und Beschreibung.

4.4 Server Sicherheitsmaßnahmen

4.4.1 Authentisierung und Autorisierung

4.4.2 Speichern von Anmeldedaten in der Datenbank

Um die Skalierbarkeit der Anwendung zu gewährleisten, werden die Anmeldedaten der Benutzer in der Datenbank gespeichert. Das hat den Vorteil, dass der Bezug zwischen Benutzer, Anmeldedaten und weiteren Informationen jederzeit gegeben ist, da keine externen Referenzen verwaltet werden müssen. Ein Nachteil dieser Vorgehensweise ist jedoch, dass jeder Datenbankbenutzer mit Lesezugriff auf das *hipsterbility*-Schema auf die Benutzerdaten zugreifen kann, wozu auch die Passwörter gehören. Um diesen Nachteil zu relativieren nutzt der *Glassfish 4* AS standardmäßig die kryptografische Hashfunktion *SHA*[23]*-2* mit 256 Bit Hashwerten (SHA-256).

Zusätzlich zu der Speicherung der Hashwerte werden die Passwörter durch den Server mit der *AES*[24] Blockchiffre verschlüsselt. Als Passwort zur Ver- und Entschlüsselung wird das *Glassfish* `Master Password` benutzt [vgl. Ora13, S. 16 f.] Leider ist diese Funktion nur schlecht dokumentiert, es wird zwar grundlegend beschrieben wie ein `JDBCRealm` eingerichtet wird, jedoch nicht die Bedeutung der einzelnen Sicherheitsparameter [vgl. Ora13, S. 50 f.] nicht im Detail erläutert.

4.5 Authentifizierung mit Java EE Security Realms

Wie in Abschnitt 4.4.2 angedeutet, wird für die Verwaltung, Autorisierung und Authentifizierung der Serverkomponente ein *JDBCRealm* eingerichtet. *Authentication Realms* bilden die Grund-

[23] Secure Hash Algorithm
[24] Advanced Encryption Standard

lage der Benutzersicherheitsmechanismen im *Glassfish* AS und ist Bestandteil von Java EE. Zu den Hauptfunktionen gehört das Authentisieren, Identifizieren und Autorisieren.

Da für die Datenspeicherung ein DBMS[25] verwendet wird, bietet es sich an dort auch die Anmeldedaten der Benutzer zu speichern.

4.5.1 JDBCRealm Datenbankschema und Entitätsklassen

Das `JDBCRealm` sieht dazu ein Schema vor, welches mindestens aus zwei Tabellen besteht (siehe Abbildung 3), eine für die Benutzer und eine für Gruppen bzw. Rollen, welche von Benutzern eingenommen werden. Die Namen der Tabellen und Spalten können bei der Konfiguration angegebene werden und sind entsprechend flexibel. Benötigt werden eine Tabelle mit Benutzernamen und Passwort (`Users`) und eine mit der Zuordnung von Gruppenname und Benutzername (`Groups`).

Bei der Verwendung der JPA könnten die zugehörigen Entity-Klassen beispielsweise aussehen, wie in Listings 1 und 2.

```
1  // [...] Imports etc.
2  @Entity(name = "Users")
3  public class UserEntity {
4
5      @Id
6      private String username;
7      private String password;
8      // [...] Getter und Setter.
9  }
```

Listing 1: Beispiel für eine UserEntity Klasse.

```
1  // [...] Imports etc.
2  @Entity(name = "Groups")
3  public class GroupEntity {
4
5      @Id
6      private int id;
7      private String name;
8
9      @ManyToOne
10     @JoinColumn(name = "username")
11     private UserEntity user;
12     // [...] Getter und Setter.
13 }
```

Listing 2: Beispiel für eine GroupEntity Klasse.

Dieses Schema hat allerdings, besonders bei der Verwendung der JPA einige Nachteile. Einerseits verhindert es das einfache Verwenden von numerischen Primärschlüsseln bei der Tabelle `Users`, da das `JDBCRealm` in der Gruppentabelle einen Benutzernamen und keine ID erwartet. Andererseits kann die selbe Gruppe einem Benutzer mehrfach zugeordnet werden, da keine Beschränkungen vorliegen.

[25] Database Management System

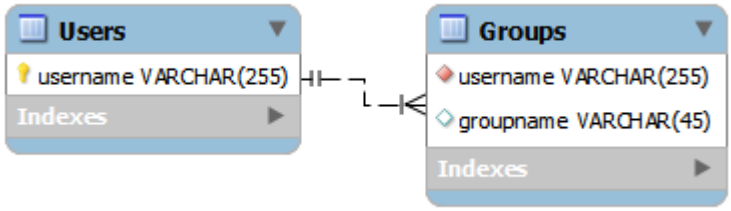

Abbildung 3: Beispielhaftes minimales Datenbankschema für ein JDBCRealm.

4.5.2 Anpassungen bei der Realisierung

Die Entitätsklassen und folglich auch das Datenbankschema wurde entsprechend der Nachteile angepasst. Die Anpassungen sind in Listings 4 und 5 dargestellt.

Um die Verwendung von numerischen Benutzer–IDs zu ermöglichen wurde eine View erstellt, die das erwartete Schema abbildet, siehe Listing 3. Außerdem wurden entsprechende Beschränkungen in den Tabellen eingeführt.

```
1 CREATE VIEW usergroup AS SELECT u.USERNAME, g.NAME FROM USER u INNER JOIN
      realmgroup g on g.USER_ID = u.ID WHERE u.ACTIVE=1;
```

Listing 3: SQL Befehl zum erstellen einer View mit Benutzername und Gruppenname

Die View hat weiterhin den Vorteil, dass die Auswahl nur aktive Benutzerkonten enthält. Wenn das Feld `ACTIVE` in der Tabelle `user` den Wert `1` hat, wird der Benutzer gelistet, hat das Feld den Wert `0`, so wird er nicht in der Auswahl erfasst. So lassen sich Benutzerkonten deaktivieren um die Anmeldung zu verhindern, müssen jedoch nicht aus der Datenbank entfernt werden.

Da der Benutzername nicht mehr der Primärschlüssel ist, wurde eine Beschränkung eingeführt, um den Benutzernamen in der Tabelle eindeutig zu halten. Außerdem darf er, nach dem Erstellen, nicht mehr geändert werden und muss einen Wert enthalten.

```
1  // [...] Imports etc.
2  @Entity(name = "User")
3  // [...]
4  public class UserEntity {
5    // [...]
6      @Id
7      @GeneratedValue(strategy= GenerationType.IDENTITY)
8      private int id;
9      @Column(unique = true, nullable = false, updatable = false)
10     private String username;
11     // [...]
12     @Column(nullable = false) @XmlTransient @JsonIgnore
13     private String password;
14     // [...] Getter, Setter und weitere Properties.
```

Listing 4: Beispiel für eine UserEntity Klasse.

Die Gruppen oder Rollentabelle wurde um Beschränkungen erweitert, die eine Mehrfachzuordnung des selben Benutzers zu einer einzigen Gruppe untersagen.

```
1  // [...] Imports etc.
2  @Entity(name = "RealmGroup")
3  @Table(uniqueConstraints = @UniqueConstraint(columnNames = {"NAME, USER_ID"}))
4  public class GroupEntity {
5      // [...]
6      @Id
7      @GeneratedValue(strategy=GenerationType.IDENTITY)
8      private int id;
9      @Column(nullable=false)
10     private String name;
11     @ManyToOne
12     private UserEntity user;
13     // [...] Getter und Setter.
14 }
```

Listing 5: Auszug aus der GroupEntity Klasse.

Wenn die nötigen JDBC[26]-Ressourcen bereits erstellt wurde, kann das `JDBCRealm` mit dem Befehl aus Listing 6 mit dem `asadmin`-Werkzeug erzeugt werden. Das Erstellen und Bearbeiten von Realms ist auch über die Weboberfläche möglich, siehe Abbildung 4.

4.6 Server Deployment

5 Realisierung des Testleiter-Clients

5.1 Realisierung des Clients für Testleiter und Administratoren

6 Weiterentwicklung der Android Bibliothek

7 Ergebnisse

8 Fazit

8.1 Weiterentwicklung

8.1.1 Integrierte Blickverfolgung

Projekte wie *Opengazer* [Zie09] zeigen eindrucksvoll, dass Eye-Tracking bzw. Gaze-Tracking auch ohne spezielles Equipment möglich ist. Mit Hilfe einer einfachen Webcam gelang dem Entwicklerteam das Verfolgen von Augenbewegungen und die Visualisierung des Fokuspunktes auf dem Computerdisplay. Die quelloffene Software benötigt für die korrekte Funktion das Markieren von Referenzpunkten im Gesicht auf dem Kamerabild, sowie eine Kalibrierung des Blickes mit Referenzpunkten auf dem Display.

Ein schwerwiegender Nachteil der aktuellen Version ist allerdings, dass der Kopf möglichst still gehalten werden und die Webcam fixiert werden muss. Selbst wenn die Software auf mobile Betriebssysteme portiert und ausgeführt werden könnte, wäre es nur schwer möglich die

[26] Java Database Connectivity

zuvor genannten Bedingungen zu erfüllen. Das Testgerät müsste fixiert werden und auch die Testperson dürfte sich beim Test möglichst wenig bewegen. Die Qualität der Messungen wäre auch bei perfekten Bedingungen fraglich, da schon kleine Messungenauigkeiten den Wert der Messpunkte stark reduzieren würden, da die Displays von mobilen Geräten sehr viel kleiner sind als bei Desktop-PCs oder Notebooks.

8.1.2 Blickerkennung mit zusätzlicher Hardware

Eine alternative zur Blickverfolgung mit der integrierten Kamera von mobilen Geräten könnte *The Eye Tribe* [The14] bieten. Diese bieten einen kompakten Eye-Tracker für $99 an. Die Hardware ist mit 1,9 x 20 x 1,9 cm Ausmaßen sehr kompakt und mit 70 g auch leicht genug für den mobilen Einsatz, zumindest an Tablets. Aktuell ist die Software nur unter Microsoft Windows lauffähig, eine Android Version befindet sich, laut Herstellerwebsite, in der Entwicklung.

Blickverfolgung auf einem Smartphone ohne externe Hardware könnte schon in naher Zukunft möglich sein. Durch die Verwendung von mehreren Kameras in der Gerätefront kann das *Amazon Fire Phone* [ama14] schon jetzt die Perspektive des Bildschirminhalts dynamisch an den Betrachtungswinkel des Benutzers anpassen. Zwar wird das Eye-Tracking vom aktuellen *Fire Phone SDK* (noch) nicht unterstützt, lediglich das Head-Tracking, jedoch ist das Smartphone erst seit kurzem in den U.S.A. verfügbar und der Verkaufsstart in Deutschland steht noch aus (stand September 2014). Welche Möglichkeiten die zusätzlichen Kameras bieten und ob sich mit ihnen eine präzise Blickverfolgung realisieren lässt wird die Zukunft zeigen.

9 Quellenverzeichnis

[ama14] amazon.de, Hrsg. *Amazon Fire Phone (Telekom)*. 2014. URL: http://www.amazon.de/gp/feature.html?ie=UTF8%5C&docId=1000819063 (besucht am 18. 09. 2014).

[And14] Android Open Source Project, Hrsg. *Get the Android SDK*. 2014. URL: http://developer.android.com/sdk/index.html (besucht am 02. 10. 2014).

[bP14] bocoup und Popcorn.js, Hrsg. *Popcorn.js*. 2014. URL: http://popcornjs.org/ (besucht am 03. 10. 2014).

[Bud14] Raluca Budiu. *Usability Testing for Mobile Is Easy*. Hrsg. von Nielsen Norman Group. 2014. URL: http://www.nngroup.com/articles/mobile-usability-testing/ (besucht am 18. 09. 2014).

[Bur14] Bill Burke. *RESTful Java with JAX-RS 2.0*. Second edition. 2014. ISBN: 9781449361341.

[Cli13] John Clingan. *Introduction to Java Platform, Enterprise Edition 7*. Hrsg. von Oracle Corporation. 2013. URL: https://www.oracle.com/technetwork/java/javaee/javaee7-whitepaper-1956203.pdf (besucht am 02. 10. 2014).

[Dar13] Javier Darriba. *5 Gründe, warum Remote Usability Testing günstiger ist als traditionelle Labor-Tests*. Hrsg. von UserZoom GmbH. 2013. URL: http://www.userzoom.de/5-grunde-warum-remote-usability-testing-gunstiger-ist-als-traditionelle-labor-tests/ (besucht am 22. 09. 2014).

[Fer14] Joe Ferner. *java: Bridge API to connect with existing Java APIs.* Hrsg. von npm. 2014. URL: https://www.npmjs.org/package/java (besucht am 02. 10. 2014).

[Joy14] Joyent, Inc, Hrsg. *About Node.js®#*. 2014. URL: http://nodejs.org/about/ (besucht am 02. 10. 2014).

[KS13] Mike Keith und Merrick Schincariol. *Pro JPA 2*. Second Edition. The expert's voice in Java. 2013. ISBN: 9781430249269.

[Ora14] Oracle, Hrsg. *Java EE 7 Technologies*. 2014. URL: http://www.oracle.com/technetwork/java/javaee/tech/index.html (besucht am 02. 10. 2014).

[Ora13] Oracle Corporation, Hrsg. *GlassFish Server Open Source Edition: Security Guide: Release 4.0*. 2013. URL: https://glassfish.java.net/docs/4.0/security-guide.pdf (besucht am 24. 09. 2014).

[Sha11] Remy Sharp. *Two videos in sync*. Hrsg. von HTML5 demos. 2011. URL: http://html5demos.com/two-videos (besucht am 03. 10. 2014).

[The14] The EyeTribe, Hrsg. *The EyeTribe*. 2014. URL: https://theeyetribe.com/ (besucht am 18. 09. 2014).

[Use14a] UserTesting Inc., Hrsg. *How It Works: UserTesting provides on-demand usability testing.* 2014. URL: http://www.usertesting.com/how-it-works (besucht am 22. 09. 2014).

[Use14b] UserTesting Inc., Hrsg. *UserTesting: Mobile Testing*. 2014. URL: http://www.usertesting.com/mobile (besucht am 22. 09. 2014).

[Use14c] UserTesting Inc., Hrsg. *UserTesting: The fastest way to get feedback*. 2014. URL: http://www.usertesting.com/ (besucht am 22. 09. 2014).

[Use13a] UserZoom GmbH, Hrsg. *Diensleistungen: Mehr als eine Software: Wir bieten alles, was Ihnen zum Erfolg verhilft.* 2013. URL: http://www.userzoom.de/dienstleistungen/ (besucht am 22. 09. 2014).

[Use13b] UserZoom GmbH, Hrsg. *Software / Research-Tools: Mobile Usability Testing*. 2013. URL: http://www.userzoom.de/software/research-capabilities/mobile-usability-testing/ (besucht am 22. 09. 2014).

[Wal11] Rick Waldron. *http://bocoup.com/weblog/html5-video-synchronizing-playback-of-two-videos/*. 2011. URL: http://bocoup.com/weblog/html5-video-synchronizing-playback-of-two-videos/ (besucht am 03. 10. 2014).

[Zie09] Piotr Zielinski. *Opengazer: open-source gaze tracker for ordinary webcams*. Hrsg. von The Inference Group. 2009. URL: http://www.inference.phy.cam.ac.uk/opengazer/ (besucht am 18. 09. 2014).

Anhang
A Server Konfiguration
A.1 Screenshots

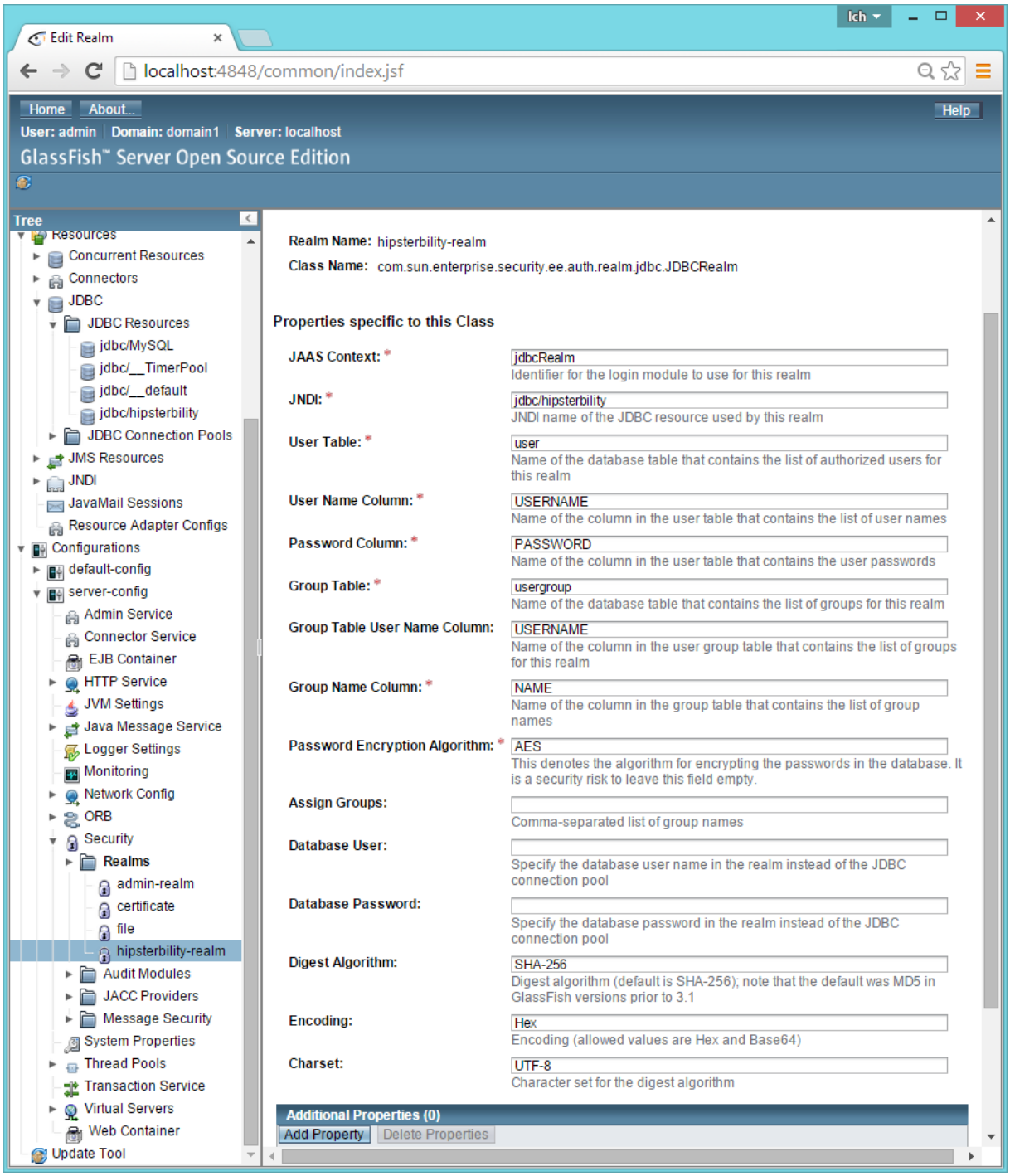

Abbildung 4: Screenshot der Security-Realm Konfiguration des Glassfish Servers

A.2 Glassfish Befehle

```
create-auth-realm --classname com.sun.enterprise.security.auth.realm.jdbc.
    JDBCRealm --property jaas-context=jdbcRealm:datasource-jndi=jdbc/
    hipsterbility:user-table=user:user-name-column=USERNAME:password-column=
    PASSWORD:group-table=usergroup:group-name-column=NAME:group-table-user-name-
    column=USERNAME:digest-algorithm=SHA-256:encoding=Hex:charset=UTF-8:
    digestrealm-password-enc-algorithm=AES hipsterbility-realm
```

Listing 6: Erstellen eines Glassfish Security Realms mit dem „asadmin“ Werkzeug.

A.3 Konfigurationsdateien

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.1">
    <persistence-unit name="HipsterbilityService" transaction-type="JTA">
        <jta-data-source>jdbc/hipsterbility</jta-data-source>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties>
            <property name="eclipselink.logging.level" value="FINE"/>
            <property name="eclipselink.ddl-generation.output-mode"
                      value="both"/>
            <property name="eclipselink.ddl-generation"
                      value="create-or-extend-tables"/>
        </properties>
    </persistence-unit>
</persistence>
```

Listing 7: persistence.xml für die Datenbankverbindung.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server
    3.1
  Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.
    dtd">

<resources>
    <jdbc-connection-pool name="jdbc_hipsterbilityPool"
                          res-type="javax.sql.ConnectionPoolDataSource"
                          datasource-classname="com.mysql.jdbc.jdbc2.optional.
                              MysqlDataSource">

        <property name="User" value="hipsterbility" />
        <property name="Port" value="3306" />
        <property name="DatabaseName" value="hipsterbility" />
        <property name="ServerName" value="localhost" />
        <property name="Url" value="jdbc:mysql://localhost:3306/hipsterbility"/>
        <property name="URL" value="jdbc:mysql://localhost:3306/hipsterbility"/>
        <property name="Password" value="hipsterbility123" />
    </jdbc-connection-pool>

    <jdbc-resource enabled="true"
                   jndi-name="jdbc/hipsterbility"
                   object-type="user"
                   pool-name="jdbc_hipsterbilityPool">
        <description />
    </jdbc-resource>
</resources>
```

Listing 8: glassfish-resources.xml zum Erzeugen eines JDBC Resource-Pools und einer JDBC Ressource

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit (entsprechend der genannten Verantwortlichkeit) selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder veröffentlicht noch einer anderen Prüfungsbehörde vorgelegt.

Datum: .....

(Unterschrift)