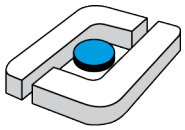

Hochschule Osnabrück
University of Applied Sciences

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

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

1 Einleitung

1.1 Motivation

1.2 Problemstellung

1.3 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
  - 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:

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

- 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:

-

4 Realisierung des Servers

4.1 Datenbank und Persistenz

4.2 RESTFul Webservice

Um die Android Bibliothek und den Testleiter-Client anzubinden wird eine REST-API[3] verwendet. Da der *Glassfish* AS[4] die Grundlage für die Serverkomponente darstellt, kommt die

[3] Application Programming Interface
[4] Application Server

JAX-RS[5] 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*[6], welches vom Ressourcenaufbau eher dem REST-RPC[7] Schema folgt.

Aktuell wird nur das JSON[8] Datenformat unterstützt. Für das (Un–)Marshalling wird serverseitig die *Jackson* (`http://jackson.codehaus.org/`) Bibliothek verwendet.

Eine Implementierung des HATEOAS[9] 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. Ein detaillierte Beschreibung der Ressourcen mit Pfaden und HTTP[10]-Methoden befindet sich in Abschnitt 4.2.2.

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

4.2.2 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. Wie auch bei den DAO's[11] in der Persistenzschicht der Serveranwendung wurden die vier grundlegenden Datenoperationen CRUD[12] implementiert. Die Zuordnung zwischen Operation und HTTP-Methoden bzw. SQL[13]-Befehlen ist in Tabelle 1 zu sehen.

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

Tabelle 1: Zuweisung der CRUD Operationen und HTTP-Methoden / SQL-Befehlen.

[5] Java API for RESTful Web Services
[6] GitHub Repository: `https://github.com/Robert-Leggett/jersey_restful_webservice`
[7] Remote Procedure Call
[8] JavaScript Object Notation
[9] Hypermedia as the Engine of Application State
[10] Hypertext Transfer Protocol
[11] Data Access Objects
[12] Create, read, update and delete
[13] Structured Query Language

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.

Daten mit Benutzerbezug (Administrator, Benutzer eingeschränkt)

GET, POST	`/users`
GET, POST, PUT, DELETE	`/users/{id}`
GET, POST	`/users/{id}/devices`
GET, POST	`/users/{id}/sessions`

Gerätedaten (Administrator)

GET, POST	`/devices`
GET, POST, PUT, DELETE	`/devices/{id}`

4.3 Server Sicherheitsmaßnahmen

4.3.1 Authentisierung und Autorisierung

4.3.2 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*[14]*-2* mit 256 Bit Hashwerten (SHA-256).

Zusätzlich zu der Speicherung der Hashwerte werden die Passwörter durch den Server mit der *AES*[15] 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.

4.4 Authentifizierung mit Java EE Security Realms

Wie in Abschnitt 4.3.2 angedeutet, wird für die Verwaltung, Autorisierung und Authentifizierung der Serverkomponente ein *JDBCRealm* eingerichtet. *Authentication Realms* bilden die Grundlage der Benutzersicherheitsmechanismen im *Glassfish* AS und ist Bestandteil von Java EE. Zu den Hauptfunktionen gehört das Authentisieren, Identifizieren und Autorisieren.

[14] Secure Hash Algorithm
[15] Advanced Encryption Standard

Da für die Datenspeicherung ein DBMS[16] verwendet wird, bietet es sich an dort auch die Anmeldedaten der Benutzer zu speichern.

4.4.1 JDBCRealm Datenbankschema und Entitätsklassen

Das `JDBCRealm` sieht dazu ein Schema vor, welches mindestens aus zwei Tabellen besteht (siehe Abbildung 1), 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[17] 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.

[16] Database Management System
[17] Java Persistence API

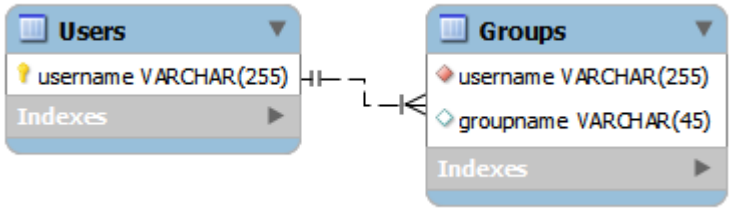

Abbildung 1: Beispielhaftes minimales Datenbankschema für ein JDBCRealm.

4.4.2 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;
```

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

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[18]-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 2.

4.5 Webservice und REST-API

5 Realisierung der Android Bibliothek

6 Realisierung des Testleiter-Clients

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 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.

[18] Java Database Connectivity

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*[19] (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.

[19] Software Development Kit

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).

[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.

[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).

[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).

[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).

[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
Screenshots

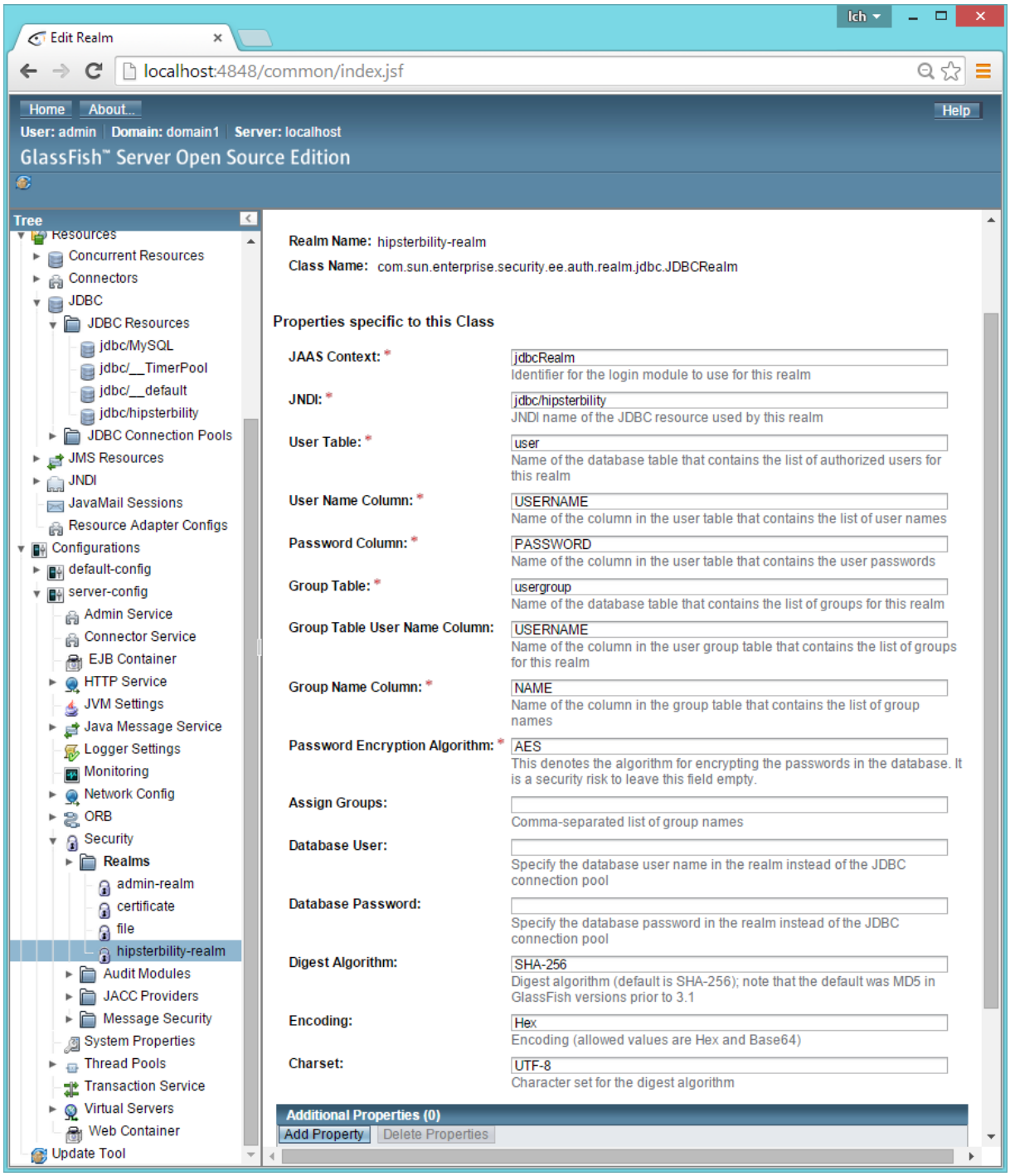

Abbildung 2: Screenshot der Security-Realm Konfiguration des Glassfish Servers

Glassfish Befehle

```
create-auth-realm --classname com.sun.enterprise.security.ee.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.

B GUI

Ein toller Anhang.

Screenshot

Unterkategorie, die nicht im Inhaltsverzeichnis auftaucht.

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)