



**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

Dieses Projekt befasst sich mit der Konzeption und Realisierung eines Usability Testing Frameworks für die Android Plattform. Die Basis bietet ein bestehender Prototyp, der systematisch erweitert wird. Kernstück ist eine Android Bibliothek, welche es ermöglicht den Bildschirminhalt, das Kamerabild der Frontkamera und weitere Daten zu erfassen. Diese Daten werden erfasst, während der Benutzer eine vorgegebene Liste von Testaufgaben nacheinander bearbeitet. Am Ende der Sitzung werden die Daten zu der zweiten Komponente, dem Server hochgeladen, wo sie der Testleiter mit einem Client abrufen kann, dritte Komponente.

Das Ergebnis soll eine Darstellung des Bildschirminhalts und das Kamerabild der Frontkamera umfassen. Ziel des Projekts ist ein Werkzeug um Remote Usability Testing Studien durchführen zu können. Diese Ziel wurde jedoch nicht in vollem Umfang erreicht.

# I Inhaltsverzeichnis

<b>II</b>	<b>Abbildungsverzeichnis</b>	<b>IV</b>
<b>III</b>	<b>Tabellenverzeichnis</b>	<b>IV</b>
<b>IV</b>	<b>Listing-Verzeichnis</b>	<b>IV</b>
<b>V</b>	<b>Abkürzungsverzeichnis</b>	<b>VI</b>
<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Bezug der Programmquellen und Dokumentation . . . . .	1
1.2	Motivation . . . . .	1
1.3	Zielsetzung . . . . .	2
1.4	Problemstellung . . . . .	2
1.5	Stand bei Projektbeginn . . . . .	2
<b>2</b>	<b>Stand der Technik</b>	<b>4</b>
2.1	Mobile Usability Labortests . . . . .	4
2.2	Remote Usability Testing Anbieter . . . . .	4
2.2.1	UserTesting . . . . .	4
2.2.2	UserZoom . . . . .	5
2.3	Technologien und Werkzeuge . . . . .	6
2.3.1	Android Bildschirmaufzeichnung . . . . .	6
2.3.2	Chromecast Screen Mirroring . . . . .	7
<b>3</b>	<b>Konzept</b>	<b>8</b>
3.1	Remote Usability Testing . . . . .	8
3.2	Architektur . . . . .	9
3.3	Evaluierung und Auswahl von Technologien . . . . .	9
3.3.1	Server . . . . .	10
3.3.2	Testleiter-Client . . . . .	11
<b>4</b>	<b>Realisierung des Servers</b>	<b>13</b>
4.1	Datenbank und Persistenz . . . . .	13
4.1.1	Konfiguration des Glassfish AS zur Verwendung einer MySQL Datenbank . . . . .	13
4.1.2	Datenmodell und Datenbankschema . . . . .	13
4.2	RESTful Webservice . . . . .	14
4.2.1	Java REST-API mit JAX-RS Annotationen und CDI . . . . .	15
4.2.2	Implementierte Ressourcen . . . . .	16
4.2.3	Aufbau von Ressourcen und Zugriffsrechte . . . . .	16
4.3	Server Sicherheitsmaßnahmen . . . . .	18
4.3.1	Speichern von Anmeldedaten in der Datenbank . . . . .	18
4.4	Authentifizierung mit Java EE Security Realms . . . . .	18
4.4.1	JDBCRealm Datenbankschema und Entitätsklassen . . . . .	18
4.4.2	Anpassungen bei der Realisierung . . . . .	20

4.4.3	Authentifizierung und Autorisierung von REST-Ressourcen . . .	21
4.5	Testen des Servers . . . . .	22
<b>5</b>	<b>Realisierung des Testleiter-Clients und der Android Bibliothek</b>	<b>23</b>
5.1	Realisierung des Clients für Testleiter und Administratoren . . . . .	23
5.1.1	Grafische Oberflächen mit dem JavaFX Scene Builder 2.0 . . . . .	23
5.1.2	Properties und Binding . . . . .	24
5.1.3	Dependency Injection mit afterburner.fx . . . . .	25
5.1.4	REST-Client . . . . .	25
5.2	Weiterentwicklung der Android Bibliothek . . . . .	26
5.3	Testen der Software . . . . .	26
<b>6</b>	<b>Ergebnisse</b>	<b>27</b>
6.1	Server . . . . .	27
6.2	Testleiter-Client und Android Bibliothek . . . . .	27
<b>7</b>	<b>Fazit</b>	<b>28</b>
7.1	Weiterentwicklung . . . . .	28
7.1.1	Integrierte Blickverfolgung . . . . .	28
7.1.2	Blickerkennung mit zusätzlicher Hardware . . . . .	29
<b>8</b>	<b>Quellenverzeichnis</b>	<b>30</b>
<b>Anhang</b>		<b>I</b>
<b>A</b>	<b>Server Konfiguration</b>	<b>I</b>
A.1	Screenshots . . . . .	I
A.2	Glassfish Befehle . . . . .	II
A.3	Konfigurationsdateien . . . . .	II
A.4	Scriptdateien . . . . .	III
A.4.1	Befehlsdateien . . . . .	III
<b>B</b>	<b>Serverkonfiguration</b>	<b>IV</b>
B.1	Diagramme . . . . .	VII
<b>C</b>	<b>Screenshots</b>	<b>VIII</b>
C.0.1	JavaFX Scene Builder . . . . .	VIII
C.0.2	Testleiter Client . . . . .	IX
C.0.3	Android Testapplikation mit Bibliothek . . . . .	XI

## II Abbildungsverzeichnis

Abb. 1	Screenshot des OmniROM Power-Menüs. . . . .	6
Abb. 2	Architektur des Frameworks. . . . .	9
Abb. 3	Beispielhaftes minimales Datenbankschema für ein JDBCRealm. . . . .	19
Abb. 4	Screenshot der Security-Realm Konfiguration des Glassfish Servers	I
Abb. 5	Datenbankschema des Hipsterbility-Servers. . . . .	VII
Abb. 6	Screenshot des JavaFX Scene Builders mit geöffneter FXML Oberfläche. . . . .	VIII
Abb. 7	Client Login Fenster. . . . .	IX
Abb. 8	Client Login Fenster, Server Einstellungen. . . . .	IX
Abb. 9	Client Benutzerverwaltung mit Testdaten. . . . .	X
Abb. 10	Android Bibliothek, Dialog zum Starten einer Testsitzung. . . . .	XI
Abb. 11	Android Bibliothek, Liste mit Test. . . . .	XI
Abb. 12	Android Bibliothek, Liste mit Aufgaben zu einem Test. . . . .	XII
Abb. 13	Android Bibliothek, erfasste Geste in einer WebView. . . . .	XIII
Abb. 14	Android Bibliothek, bei der Aufzeichnung erfasste Geste und zweiter Berührungspunkt. . . . .	XIII

## III Tabellenverzeichnis

Tab. 1	Tabellen und Entitäten der Persistenzschicht. . . . .	14
Tab. 2	Zuweisung der CRUD Operationen und HTTP-Methoden und SQL-Befehlen. . . . .	16
Tab. 3	Einige REST-Ressourcen mit HTTP-Methoden, Pfad, Benutzerrolle und Beschreibung. . . . .	17

## IV Listing-Verzeichnis

1	Beispiel für eine UserEntity Klasse. . . . .	19
2	Beispiel für eine GroupEntity Klasse. . . . .	19
3	SQL Befehl zum erstellen einer View mit Benutzername und Gruppenname . . . . .	20
4	Beispiel für eine UserEntity Klasse. . . . .	20
5	Auszug aus der GroupEntity Klasse. . . . .	21
6	glassfish-web.xml Mapping von SecurityRealm Rollen. . . . .	22
7	Erstellen eines Glassfish Security Realms mit dem „asadmin“ Werkzeug.	II
8	glassfish-resources.xml zum Erzeugen eines JDBC Resource-Pools und einer JDBC Ressource . . . . .	II
9	glassfish-web.xml JSP Konfiguration und Mapping von SecurityRealm Rollen. . . . .	II
10	glassfish-create.txt Befehle zu erzeugen von Ressourcen. . . . .	III
11	glassfish-delete.txt Befehle zum Löschen von Ressourcen. . . . .	III
12	glassfish-create-resources.bat . . . . .	III
13	glassfish-delete-resources.bat . . . . .	IV
14	glassfish-create-resources.sh . . . . .	IV

15	glassfish-delete-resources.sh . . . . .	IV
16	persistence.xml (Datenbankverbindung) . . . . .	IV
17	web.xml (Servlet Konfiguration) . . . . .	IV

## V Abkürzungsverzeichnis

<b>ADB</b>	Android Debug Bridge
<b>AES</b>	Advanced Encryption Standard
<b>API</b>	Application Programming Interface
<b>AS</b>	Application Server
<b>CDI</b>	Contexts and Dependency Injection
<b>CRUD</b>	Create, read, update and delete
<b>CSS</b>	Cascading Style Sheets
<b>DAO</b>	Data Access Object
<b>DBMS</b>	Database Management System
<b>GPL</b>	GNU General Public License
<b>GUI</b>	Graphical User Interface
<b>HATEOAS</b>	Hypermedia as the Engine of Application State
<b>HTML</b>	Hypertext Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>JAX-RS</b>	Java API for RESTful Web Services
<b>JAAS</b>	Java Authentication and Authorization Service
<b>JDBC</b>	Java Database Connectivity
<b>JDK</b>	Java SE Development Kit
<b>JNI</b>	Java Native Interface
<b>JPA</b>	Java Persistence API
<b>JSF</b>	JavaServer Faces
<b>JSON</b>	JavaScript Object Notation
<b>JSP</b>	JavaServer Pages
<b>MTP</b>	Media Transfer Protocol
<b>MVC</b>	Model View Controller
<b>MVP</b>	Model View Presenter
<b>ORM</b>	Object-relational mapping
<b>POJO</b>	Plain Old Java Object
<b>REST</b>	Representational State Transfer
<b>RPC</b>	Remote Procedure Call
<b>SDK</b>	Software Development Kit
<b>SHA</b>	Secure Hash Algorithm
<b>SQL</b>	Structured Query Language
<b>URL</b>	Uniform Resource Locator
<b>USB</b>	Universal Serial Bus
<b>WYSIWYG</b>	„What You See Is What You Get“
<b>XML</b>	Extensible Markup Language

# 1 Einleitung

## 1.1 Bezug der Programmquellen und Dokumentation

Die Programmquellen der entwickelten, bzw. weiterentwickelten Software ist in dem folgenden *GitHub*-Repository abrufbar:

<https://github.com/Chromdioxyde/hipsterbility>

Als Versionsverwaltung wird *Git*<sup>1</sup> eingesetzt. Mit dem Befehl

`git clone https://github.com/Chromdioxyde/hipsterbility.git` kann eine lokale Kopie erstellt werden. Alternativ kann der Inhalt des Repositories auf der o. g. Webseite auch als ZIP-Datei heruntergeladen werden. Die Quellen der Programme liegen im *IntelliJ IDEA*<sup>2</sup> Projektformat vor. Für die Entwicklung wurde *IntelliJ IDEA Ultimate 13.1.5* verwendet.

## 1.2 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 1.5).

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

---

<sup>1</sup>Git Webseite: <http://git-scm.com/>

<sup>2</sup>IntelliJ IDEA Website: <https://www.jetbrains.com/idea/>



## 1.3 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. Die Serverkomponente wurde ausgelagert, für den Fall, dass ein Server zur Verfügung steht, bzw. z. B. von einer Bildungseinrichtung bereitgestellt wird.

## 1.4 Problemstellung

Wie in Abschnitt 2 näher erläutert wird, gibt es zwar kommerzielle Anbieter für das Remote Usability Testing, jedoch beschränkt sich deren Angebot entweder auf mobile Websites oder ist mit hohen Kosten verbunden. Auch die verfügbaren Werkzeuge bieten keine zufriedenstellende Lösung für das entfernte Testen.

Weiterhin besteht zwar die Möglichkeit den Bildschirm von Android Geräten aufzuzeichnen und Interaktionen darzustellen, jedoch wird dafür entweder ein zusätzlicher PC benötigt, oder es besteht keine einfache Möglichkeit diese Aufnahmen auswertbar zu machen und weiterzugeben.

Neben den Bildschirmaufgaben müssen auch Aufgaben und Anweisungen an den Benutzer übermittelt, sowie zusätzliche Metadaten gesammelt werden, um später statistische Aussagen machen zu können.

Für einen Teil dieser Probleme bestehen bereits Lösungsansätze, auf welche im nächsten Abschnitt verwiesen wird.

## 1.5 Stand bei Projektbeginn

Dieses Projekt basiert auf einer Arbeit, welche zusammen mit Oliver Erxleben<sup>3</sup> im Modul *Mensch-Maschine-Kommunikation* (Wintersemester 2013/2014) des Informatik–

---

<sup>3</sup>(oliver.erxleben@hs-osnabrueck.de)

Masterstudiengangs *Verteilte und mobile Anwendungen* erstellt wurde. Die Dokumentation *Entwicklung eines Usability-Test-Frameworks für Android* [EH14] wurde als Prüfungsleistung für das zuvor genannte Modul eingereicht. Die Ergebnisse zum Stand der Abgabe können aus dem Git-Repository von Oliver Erxleben unter der folgenden Webadresse abgerufen werden:

[https://github.com/olivererxleben/hipsterbility/releases/tag/v1.0\\_semester\\_final](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 Representational State Transfer (REST)-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:

- 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 (siehe Abschnitt 4).

## 2 Stand der Technik

### 2.1 Mobile Usability Labortests

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] Dies ist ein Beispiel für gängige Labortests. Testmethoden für Labortests werden an dieser Stelle nicht weiter behandelt. Im nächsten Abschnitt werden zwei Anbieter für Remote Usability Tests vorgestellt.

### 2.2 Remote 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 während 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, z. B. nach der *Thinking Aloud* Methode, 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 Technologien und Werkzeuge

### 2.3.1 Android Bildschirmaufzeichnung

Eine der Neuerungen, welche mit Android 4.4 „KitKat“ eingeführt wurde, ist die Möglichkeit den Bildschirminhalt in Form eines Videos aufzuzeichnen. Die Aufzeichnung wird über einen PC gestartet, der mit dem Android Gerät über die Android Debug Bridge (ADB) [And14a] per Universal Serial Bus (USB)-Kabel oder kabellos verbunden ist. Die Aufzeichnung wird mit dem Befehl `adb shell screenrecord` gestartet werden. Soll ein Teil einer Applikation nicht aufgezeichnet werden, so kann der Entwickler die Aufnahme mit `SurfaceView.setSecure()` verhindern. Die Aufzeichnung wird im MP4-Format auf dem Gerät gespeichert und lässt sich von dort abrufen, z. B. mit der ADB oder über eine USB Media Transfer Protocol (MTP). [Vgl. And14b]

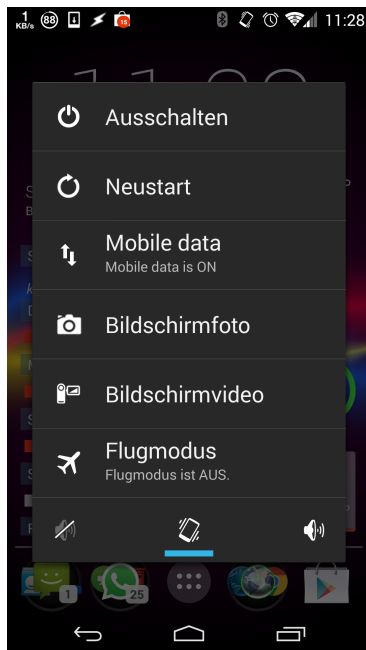


Abbildung 1: Screenshot des OmniROM Power-Menüs.

ADB ist ein Debugging-Werkzeug, welches als Bestandteil des Android Software Development Kit (SDK) [And14c] ausgeliefert wird. Mit ihm lassen sich Befehle direkt auf Shell des Geräts ausführen. Vor Android 4.4 war es bereits möglich mit dem Befehl `adb shell screencap -p /sdcard/<name>.png` Bildschirmfotos zu erstellen [vgl. Ran13]. Diese Möglichkeit besteht seit Android 4.0 „Ice Cream Sandwich“ (ICS).

Die beiden genannten Möglichkeiten für Aufzeichnungen des Bildschirminhalts benötigen jedoch einen per ADB verbundenen PC.

Um diese Befehle direkt auf dem Gerät aus einer Applikation heraus auszuführen, wird der Root-Zugriff auf dem Gerät benötigt [vgl. EH14, S. 22].

Alternativ bieten einige Geräte und sog. „Custom-ROMs“ die Möglichkeit Bildschirmfotos und -videos direkt aus einem Menü heraus oder per Tastenkombination zu erstellen. Abbildung 1 zeigt beispielhaft das Power-Menü von *OmniROM*<sup>4</sup>.

Eine Möglichkeit um den Bildschirminhalt auf einem zusätzlichen Display anzuzeigen wird im nächsten Abschnitt kurz erläutert.

<sup>4</sup>OmniROM Webseite: <http://omnirom.org/>

### 2.3.2 Chromecast Screen Mirroring

Der Google *Chromecast* ist ein Streaming-Client, welcher z. B. an Monitoren oder Fernsehern angeschlossen werden kann. Nach der ersten Einrichtung kann er u. a. über Android Geräte im selben Netzwerk gesteuert werden. Applikationen mit *Chromecast* Unterstützung können Medien auf diesem Weg Inhalte direkt auf einem Bildschirm darstellen. [Vgl. Goo14a] Neben dem Streaming von Medieninhalten aus dem Internet unterstützt die aktuelle *Chromecast* Applikation für Android [Goo14b] auch das Übertragen des Bildschirminhalts mit sehr geringer Latenz. Diese Funktion ist allerdings nicht für alle Geräte verfügbar. Eine Aufzeichnung der Übertragung wird aktuell nicht unterstützt.

Auch wenn es andere Lösungen gibt, um den Bildschirminhalt eines mobilen Geräts zu übertragen, *Chromecast* ist mit 35 €<sup>5</sup> einer der preiswertesten Wege, bei dem nur ein Fernseher oder Monitor benötigt wird. Besonders spannend sind die Möglichkeiten für Labortests, denn die Kamera, welche den Bildschirm abfilmt könnte entfallen und der Testbenutzer kann sich frei im Raum, bzw. in der Reichweite des kabellosen Netzwerks, bewegen.

---

<sup>5</sup>Preisangabe des Herstellers, Stand: Oktober 2014. Webseite: <https://www.google.de/chrome/devices/chromecast/>

## 3 Konzept

### 3.1 Remote Usability Testing

Remote Usability Testing kann in zwei Varianten durchgeführt werden, moderiert oder nicht moderiert. Bei der nicht moderierten Variante haben Studienteilnehmer bzw. Testbenutzer nur die ihnen gegebene Aufgabenstellung und Informationen zur Verfügung. Diese Testmethode ist beispielsweise beim Testen von Webseiten gängig. Mittlerweile gehört die Untersuchung von mobilen Webseiten zu dem Angebot einiger Dienstleister (siehe Abschnitt 2.2).

Um eine solche Studie durchzuführen werden üblicherweise die folgenden vier Schritte benötigt [Vgl. Sou10]:

**Definieren der Studie:** Zusammenstellen von Aufgaben und Fragen, welche von den Teilnehmern bearbeitet werden sollen.

**Rekrutieren von Teilnehmern:** Testpersonen werden entweder direkt oder durch Dienstleister kontaktiert und gebeten an der Studie teilzunehmen.

**Verschicken von Einladungen und Starten des Tests:** Wenn die zuvor genannten Schritte erfüllt sind, kann die Studie gestartet werden und die Teilnehmer werden z. B. per E-Mail benachrichtigt.

**Analysieren der Ergebnisse:** Das Auswerten der Testergebnisse wird häufig von Werkzeugen unterstützt, die von Dienstleistern bereitgestellt werden.

Die Dauer der Tests sollen drei bis 15 Minuten umfassen und es sollten nicht mehr als drei bis fünf Testaufgaben gestellt werden, da eine zu hohe Dauer und zu viele Aufgaben die Erfolgsrate bei der Erfüllung der Aufgaben reduzieren und für eine höhere Ausstiegsrate sorgen.[Vgl. Sou10]

Diese Methode soll hier durch Software zum nativen Testen von Android Applikationen nachgebildet werden.

Der nächste Abschnitt stellt die geplante Architektur der Anwendung dar.

### 3.2 Architektur

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

Der Aufbau der Komponenten ist in Abbildung 2 dargestellt. Die Kommunikation zwischen den Komponenten erfolgt per Hypertext Transfer Protocol (HTTP) und ein REST - Application Programming Interface (API). Der Server nutzt zur Datenhaltung eine Datenbank, auf welche durch die Geschäftslogik zugegriffen wird.

Der Administrations-Client dient zum Verwalten der Tests, Aufgaben und Benutzer, sowie zum Einsehen von Ergebnissen bereits durchgeführter Testsitzungen. Er besitzt keine eigene Datenhaltung und erhält sämtliche Daten vom Server über einen REST-Client.

Die Android Bibliothek kommuniziert auch über einen REST-Client mit dem Server. Um Testsitzungen auch ohne Internetverbindung oder allgemein der Verbindung zum Server durchführen zu können, speichert die Android Bibliothek Tests, Aufgaben und die gesammelten Daten von Testsitzungen lokal zwischen. Diese werden bei der nächsten Verbindung mit dem Server an diesen übertragen.

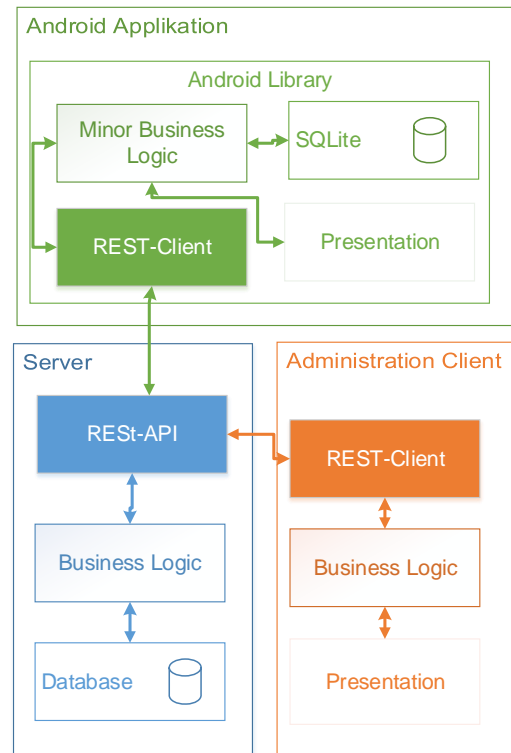


Abbildung 2: Architektur des Frameworks.

### 3.3 Evaluierung und Auswahl von Technologien

Im Gegensatz zu dem Ursprünglichen Projekt (siehe Abschnitt 1.5) liegt hier nicht der Fokus 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 Systemvoraussetzungen [vgl. And14c] geht außerdem hervor, dass für das Entwickeln ein Computer vorhanden sein muss, welcher als Betriebssystem ein halbwegs aktuelles Microsoft Windows, Max OS X oder Linux besitzt, auf dem mindestens das Java SE Development Kit (JDK) 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 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.

Zwar existiert ein Bridge-API um Java Code in JavaScript verwenden zu können. Die Aufrufe erfolgen intern über Java Native Interface (JNI) 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. Ora14a]

Als Java EE 7 Implementierung wird der *Glassfish 4.1* (<https://glassfish.java.net/>) Application Server (AS) 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 JavaServer Pages (JSP) bzw. JavaServer Faces (JSF)/Facelets basiert. Eine Neuerung in Java EE 7 ist die Hypertext Markup Language (HTML) 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 Framework 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]

Um keine Brüche in der Programmiersprache zu haben und um die bekannte Umgebung von Java/Android Entwicklern nicht zu weit zu verlassen wurde nach einer Alternative auf Basis von Java SE gesucht. Das Swing Oberflächenframework bekommt mit der Einführung von Java 8 Konkurrenz durch JavaFX, welches zwar grundsätzlich schon seit 2007 verfügbar ist, jedoch bisher nur separat vertrieben wurde. Im Vergleich zum angestaubten Swing Framework bietet JavaFX u. a. moderne Oberflächen auf Extensible Markup Language (XML)-Basis, eine umfangreiches Medienframework und hardwarebeschleunigte 3D Grafik. Seit dem 18.03.2014 ist es zudem Bestandteil des JDK 8 und auch in Java SE 8 enthalten [Vgl. Dea14, S. xxiv-xxvi]. Besonders aufgrund der neuen und umfangreichen Medienfunktionen wird beim Testleiter Client auf JavaFX 8 gesetzt.

Zunächst wird jedoch im folgenden Abschnitt beschrieben, wie der Server realisiert wird, da dieser das Grundgerüst des Frameworks darstellt.

## 4 Realisierung des Servers

### 4.1 Datenbank und Persistenz

Um die gesammelten Daten von den Testsessions, Metadaten, Tests und Aufgaben verwalten und speichern zu können, wird ein Database Management System (DBMS) verwendet. Bei der Entwicklung wurde weitgehend darauf geachtet keine herstellerspezifischen Funktionen und Eigenschaften zu verwenden. Für die Entwicklung wird eine *MySQL*-Datenbank verwendet, welche laut Aussage des Anbieters „The world’s most popular open source database“ [Ora14e] sei. Die Datenbank kommt in der quelloffenen „Community Edition“ (GNU General Public License (GPL)-Lizenz) in der Version 5.7 zum Einsatz.

Als Abstraktionsschicht wird die Java Persistence API (JPA) verwendet, als Provider *EclipseLink*. Letzterer ist der JPA Standardprovider des *Glassfish* AS.

#### 4.1.1 Konfiguration des Glassfish AS zur Verwendung einer MySQL Datenbank

Mit dem kostenlos verfügbarem Werkzeug *MySQL Workbench* [Ora14b] wurden zuerst ein Datenbankschema *hipsterbility* und ein Benutzer *hipsterbility* angelegt. Dieser Benutzer erhält volle Zugriffsrechte auf das erstellte Schema. Da der *Glassfish*-Server und die *MySQL*-Datenbank auf dem selben System betrieben wurden, erhält der Benutzer nur lokale Anmelderechte. Bei Fernanmeldungen sollte ein starkes Passwort verwendet werden.

Im nächsten Schritt wird der *MySQL*-Java Database Connectivity (JDBC) Treiber in das entsprechende Verzeichnis des *Glassfish*-Server kopiert. Dieser Schritt ist in der Dokumentation [Ora14c] ausführlich beschrieben.

#### 4.1.2 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-Entitätsklassen (Entity). Dies sind mit JPA-Annotationen versehene Plain Old Java Object (POJO)-Klassen sind [vgl. KS13, S. 17-19].

Das resultierende Datenbankschema (siehe Abbildung 5) wird durch den Object-relational mapping (ORM)-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 4.3.1)

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 Benutzer
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	<i>DeviceClass</i> 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 Persistenzschicht.

## 4.2 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 Java API for RESTful Web Services (JAX-RS) 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*<sup>6</sup>, welches vom Ressourcenaufbau eher dem REST-Remote Procedure Call (RPC) Schema folgt.

Aktuell wird nur das JavaScript Object Notation (JSON) Datenformat unterstützt. Für das (Un-)Marshalling wird serverseitig die *Jackson* (<https://github.com/FasterXML/jackson>) Bibliothek verwendet.

Eine Implementierung des Hypermedia as the Engine of Application State (HATEOAS) 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 und Ressourcen bekannt ist. Eine detaillierte Beschreibung der Ressourcen mit Pfaden und HTTP-Methoden befindet sich in Abschnitt 4.2.2.

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

Die Serveranwendung besteht insgesamt aus den folgenden drei Schichten:

1. Data Access Layer
2. Service Layer
3. Resource Layer

Der Data Access Layer abstrahiert die Datenbankbindung und implementiert die Create, read, update and delete (CRUD)-Methoden für den Datenzugriff. Für jeden Entitätentyp wurde ein eigenes Data Access Object (DAO) erstellt, lediglich die Verwaltung von *FileEntity* Subklassen wurde in einem DAO zusammengefasst, da sich diese Objekte in ihrer Struktur sehr ähnlich sind. Der Service Layer nutzt ein oder mehrere DAO's für die Persistenz und abstrahiert diese wiederum für den Resource Layer, welcher die REST-API bildet und selbst auch ein oder mehrere Services nutzt.

Die DAO und Service Objekte werden per Contexts and Dependency Injection (CDI) in die Objekte der jeweils nächsten Schicht injiziert. Dies eliminiert einen Teil der Objektverwaltung zur Laufzeit, da sich der Container derer annimmt. Der Container scannt die Klassen automatisch nach entsprechenden Annotationen ab, sofern die Bindings nicht manuell spezifiziert werden. Neben fast allen Java-Klassen lassen sich u. a. auch Session Beans, Java EE Ressourcen und Kontext Objekte injizieren. Außerdem besteht so

---

<sup>6</sup>GitHub Repository: [https://github.com/Robert-Leggett/jersey\\_restful\\_webservice](https://github.com/Robert-Leggett/jersey_restful_webservice)

die Möglichkeit verschiedene Implementierungen für ein Interface bereitzustellen, von denen eine während der Laufzeit ausgewählt wird. [Vgl. Jun13, S. 497 ff.]

Bei den DAO und Services wurde die `@Singleton` Annotation gewählt, wodurch diese als Singleton Session Beans deklariert werden [vgl. Ora13a]. Dies wurde gewählt, weil die Objekte keinen Zustand speichern – `@Stateless` hätte dies auch erfüllt – und der Dienst insgesamt nicht auf die Bewältigung von einer Großen Anzahl von Anfragen ausgelegt ist.

#### 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 (siehe Abschnitt 4.1).

Wie auch bei den DAO's in der Persistenzschicht der Serveranwendung wurden die vier grundlegenden Datenoperationen CRUD implementiert. Die Zuordnung zwischen Operation und HTTP-Methoden bzw. Structured Query Language (SQL)-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.

#### 4.2.3 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 Ressourcen mit den implementierten HTTP-Methoden aufgelistet. Die Pfadangabe erfolgt relativ zur Basis-Uniform Resource Locator (URL). Angaben in geschweiften Klammern sind Pfad-Parameter, welchen im Aufruf ein Wert zugewiesen ist.

<b>Rollen</b>	<b>Methode</b>	<b>Pfad</b>	<b>Beschreibung</b>
USER	GET	/users	Abrufen der Daten des angemeldeten Benutzers
ADMIN	GET	/users	Liste aller Benutzer
USER	PUT	/users	Angemeldeten Benutzer aktualisieren
ADMIN	PUT	/users/{id}	Benutzer mit der ID <i>id</i> aktualisieren
ALL, ADMIN	POST	/users	Neuen Benutzer erstellen
ADMIN	GET	/users/{id}	Ausgabe des Benutzers mit der ID <i>id</i>
ADMIN	DELETE	/users/{id}	Löschen des Benutzers mit der ID <i>id</i>
USER	GET	/devices	Liste mit Geräten des Benutzers
ADMIN	GET	/devices	Liste mit allen Geräten
ADMIN	GET	/devices/{id}	Abrufen Gerät mit der ID <i>id</i>
ADMIN	DELETE	/devices/{id}	Löschen des Geräts mit der ID <i>id</i>
ADMIN	PUT	/devices/{id}	Aktualisieren des Geräts mit der ID <i>id</i>

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

Tabelle 3 zeigt den Aufbau einiger Ressourcen mit zugelassenen Benutzerrollen, Pfaden und Parametern. Diese Ressourcen sollen als Beispiel dienen, für die verschiedenen Ebenen des Ressourcenaufbaus, denn je nach Benutzerrolle verhalten sich bestimmte Ressourcen anders als bei anderen Rollen.

Für Ressourcen die Objekte mit Objektlisten als Klassenvariablen besitzen wurde eine mehrstufige Hierarchie aufgebaut, nach dem Schema `/resource/{id}/subresource/{id}`.

Wie diese Ressourcen und die Anmeldedaten der Benutzer gesichert werden, ist im nächsten Abschnitt beschrieben.



## 4.3 Server Sicherheitsmaßnahmen

### 4.3.1 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 *Secure Hash Algorithm (SHA)-2* mit 256 Bit Hashwerten (SHA-256).

Zusätzlich zu der Speicherung der Hashwerte werden die Passwörter durch den Server mit der *Advanced Encryption Standard (AES)* Blockchiffre verschlüsselt. Als Passwort zur Ver- und Entschlüsselung wird das *Glassfish Master Password* benutzt [vgl. Ora13b, 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. Ora13b, S. 50 f.] nicht im Detail erläutert.

## 4.4 Authentifizierung mit Java EE Security Realms

Wie in Abschnitt 4.3.1 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.

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

### 4.4.1 JDBCRealm Datenbankschema und Entitätsklassen

Das *JDBCRealm* sieht zum Speichern der Benutzerdaten und Rollen 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 angegeben 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
7       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 = "
11      username")
12    private UserEntity user;
13    // [...] Getter und
14      Setter.
15 }

```

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.

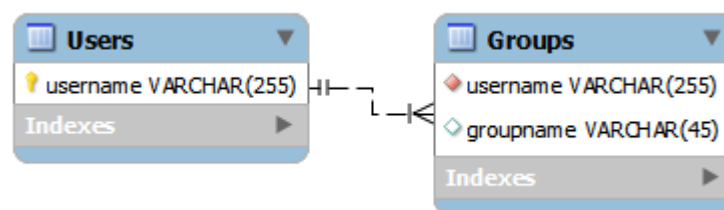


Abbildung 3: Beispielhaftes minimales Datenbankschema für ein JDBCRealm.

### 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 realmgroupp g on g.USER_ID = u.ID WHERE u.ACTIVE=1;
```

Listing 3: SQL Befehl zum erstellen einer View mit Benutzernamen 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 Benutzernamen 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-Ressourcen bereits erstellt wurde, kann das JDBCRealm mit dem Befehl aus Listing 7 mit dem `asadmin`-Werkzeug erzeugt werden. Das Erstellen und Bearbeiten von Realms ist auch über die Weboberfläche möglich, siehe Abbildung 4. Der folgende Abschnitt beschreibt die Sicherung der REST-Ressourcen vor unzulässigen Zugriffen.

#### 4.4.3 Authentifizierung und Autorisierung von REST-Ressourcen

Für die Autorisierung und Authentifizierung wird die *Glassfish* Implementierung der Java Authentication and Authorization Service (JAAS) API genutzt. Wie zuvor beschrieben, sind Rollen und Benutzer in der Datenbank hinterlegt. Für die Zuweisung der Rollen aus der Datenbank zu den Rollen, die im Webdienst verwendet werden dient die Konfigurationsdatei `glassfish-web.xml`. Listing 6 zeigt den relevanten Teil der Konfiguration, indem die SecurityRealm Gruppen den Rollen des Webservice zugewiesen werden (im Anhang ist die vollständige Konfigurationsdatei zu finden).

Anschließend wurden in der `web.xml` Servlet-Konfigurationsdatei (siehe Anhang Listing 17) die SecurityConstraints mit den erlaubten Rollen und URL Pfaden eingetragen.

Die verwendete Authentisierung ist die HTTP *Basic Authentication*, bei der werden Benutzername und Passwort Base 64 kodiert im Header übertragen [vgl. Bur14, S. 216 f.].

In den Ressourcenklassen werden Pfade und Methoden mit Annotationen versehen, die zur Verwaltung der Zugriffsrechte dienen. Um die Funktion in Jersey zu aktivieren muss sie mit `register(RolesAllowedDynamicFeature.class)` in der Konfiguration der Anwendung aktiviert werden. Danach stehen die Annotationen `@RolesAllowed`, `@PermitAll` und `@DenyAll`. Die erste der drei legt die erlaubten Rollen für authentifizierte Benutzer fest. `@PermitAll` erlaubt den Zugriff für alle Rollen und `@DenyAll` erlaubt nur den Zugriff aus der Anwendung selbst. [Vgl. Bur14, S. 219 ff.]

```
1    <security-role-mapping>
2        <role-name>ADMIN</role-name>
3        <group-name>ADMIN</group-name>
4    </security-role-mapping>
5    <security-role-mapping>
6        <role-name>USER</role-name>
7        <group-name>USER</group-name>
8    </security-role-mapping>
```

Listing 6: glassfish-web.xml Mapping von SecurityRealm Rollen.

Im folgenden Abschnitt wird kurz beschrieben, wie der Server getestet wurde.

## 4.5 Testen des Servers

Das Testen des Servers ist aufgrund der Dependency Injection nicht einfach mit JUnit testbar, da JUnit keinen Servlet Container bereitstellt. Aus diesem Grund wurden während der Entwicklung verschiedenen Integrationstests durchgeführt, teilweise auch mit JUnit. Auf die Erstellung einer vollständigen Testsuite der Komponenten wurde aus Zeitgründen verzichtet. Das dies ein Fehler war zeigte sich im Verlauf der Entwicklung, da z. B. führten Veränderungen an Entitätsklassen zu Fehlermeldungen, dass die Entity dem JPA `EntityManager` nicht bekannt sei. Wie sich später herausstellte wurde durch das erneute Deployment der Anwendung die `EntityManagerFactory` im Glassfish AS nicht geschlossen und neu erstellt, sondern die vorherige Version weiterverwendet, was zu schwer erklärbaren Fehlermeldungen führte.

Die REST-API wurde Anfangs mit der Browsererweiterung *Postman*<sup>7</sup> für den Google Chrome Browser, später mit Unit-Tests im Testleiter-Client.

<sup>7</sup>Postman im Chrome Webstore: <https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm>

## 5 Realisierung des Testleiter-Clients und der Android Bibliothek

### 5.1 Realisierung des Clients für Testleiter und Administratoren

Der Testleiter Client wurde auf Basis von JavaFX mit dem JDK 8 erstellt. JavaFX ist seit Java 8 Bestandteil der Java SE Laufzeitumgebung und unterstützt viele Neuerungen wie Lambda-Methoden und Property-Bindings. Neben der einfachen Graphical User Interface (GUI) Entwicklung, wird auch das Rendern von dreidimensionalen Objekten und Szenen mit Hardwarebeschleunigung unterstützt. Bei JavaFX bietet außerdem Zahlreiche Methoden an um Animationen zu gestalten und Medien wiederzugeben und zu manipulieren.

Das Hauptnachsschlagewerk für die Implementierung wird das Buch *JavaFX 8: Introduction by example* von Dea [Dea14].

Nachfolgend wird die Erstellung von grafischen Oberflächen in JavaFX kurz beschrieben.

#### 5.1.1 Grafische Oberflächen mit dem JavaFX Scene Builder 2.0

Ähnlich der XML Layouts bei der Android Plattform, bietet JavaFX die Möglichkeit Oberflächen deklarativ und außerhalb des Quellcodes zu definieren. Dies geschieht durch XML basierte FXML-Dateien. Um diese nicht manuell erstellen zu müssen, stellt Oracle das Werkzeug *Scene Builder* [Ora14d] zur Verfügung, mit welchen sich Oberflächen im „What You See Is What You Get“ (WYSIWYG) Stil erstellen lassen.

Abbildung 6 zeigt, den *Scene Builder* mit einer geöffneten FXML-Datei. Wichtige Elemente für das Erstellen von Oberflächen sind die Zuweisung einer Controller Klasse, die Vergabe der `fx:id` und bei Schaltflächen oder interaktiven GUI-Elementen die `onAction` Methode.

Die Möglichkeit eine Controller-Klasse zu vergeben zeigt, dass die Verwendung des Model View Controller (MVC) oder Model View Presenter (MVP) Entwurfsmuster vorgesehen, bzw. leicht möglich, ist. Bei der Implementierung wurde das MVP Entwurfsmuster umgesetzt, welches sich dadurch auszeichnet, dass möglichst wenig Programmlogik im View, der Oberfläche, verwendet wird. Der Presenter beherbergt einen Großteil der Programmlogik und stellt das Bindeglied zwischen Model und View dar. [Vgl. Dea14, S. 74]

Wenn ein GUI-Element eine zugewiesene `fx:id` (Abbildung 6 rechts oben) besitzt, so kann es mit der `@FXML` Annotation an einer Objektvariable in der zugewiesenen Controller-Klasse direkt injiziert werden. Dies erleichtert die Entwicklung, da diese Objekte nicht mehr manuell instanziiert und der Oberfläche zugewiesen werden müssen. Das injizieren mit der `@FXML` Annotation ist auch bei Methoden möglich, denen eine `On Action` Methode vergeben wird. Dadurch entfällt das manuelle Zuweisen eines `ActionListeners`.

### 5.1.2 Properties und Binding

Das neu eingeführte Property-Binding ermöglicht das Verknüpfen von GUI-Elementen, teilweise auch als *Node*, *Control* oder *View* bezeichnet, mit Properties, welche in der einfachsten Form Wrapper für primitive Objekte darstellen.

Nachfolgend einige Beispiele für Property-Klassen (`javafx.beans.property`) in JavaFX:

- `SimpleBooleanProperty`
- `ReadOnlyBooleanWrapper`
- `SimpleIntegerProperty`
- `ReadOnlyIntegerWrapper`
- `SimpleStringProperty`
- `ReadOnlyStringWrapper`

Diese Beispiele zeigen einige der Property Klassen, welche in zwei Gruppen eingeteilt werden können, Properties mit Lese-/Schreibzugriff und schreibgeschützte Properties. Der Hauptvorteil dieser Wrapper-Klassen liegt darin, dass sie sowohl die Schnittstelle `Observable`, als auch `ObservableValue` implementieren und sich mit `addListener()` `ChangeListener` und `InvalidationListener` registrieren lassen. Dadurch müssen die Werte nicht durch Polling abgefragt werden und die Listener werden bei Ereignissen benachrichtigt. Die Implementierung der Listener ist durch Java-Klassen, anonyme Klassen oder Lambda-Ausdrücke möglich. [Vgl. Dea14, S. 75-79]

Ein weitere Eigenschaft von Properties sind Bindings. Dadurch lassen sich zwei oder mehrere Werte synchron halten, indem Properties an GUI Elemente gebunden werden. Einfache Bindings, bei denen nur die Werte synchronisiert werden, gibt es in zwei Varianten, unidirektional und bidirektional. Beim unidirektionalen Binding mit `bind()` wird die Property aktualisiert, an welcher `bind()` aufgerufen wurde. Bei bidirektionaler Wertebindung mit `bindBidirectional()` werden die Werte in beide Richtungen

synchronisiert. Neben der einfachen Wertebindung gibt es noch weitere Bindings die sich wie boolsche Operatoren oder mathematische Funktionen verhalten. [Vgl. Dea14, S. 79-81]

Diese werden hier jedoch nicht weiter behandelt.

### 5.1.3 Dependency Injection mit *afterburner.fx*

*afterburner.fx* [Bie14] ist ein minimalistisches Framework, bestehend aus zwei Java Klassen, welches die beiden Prinzipien *Convention over Configuration* und *Dependency Injection* in JavaFX Anwendungen ermöglicht. Durch das Befolgen eines vorgegebenen Dateischemas und Packageaufbaus können Views praktisch ohne Code erstellt werden, einzig durch eine FXML-Datei und eine Klasse die von `FXMLView` erbt.

Das Vorgegebene Schema sieht für „MyClass“ etwa wie folgt aus:

```
/
├─ MyClassPresenter.java
├─ MyClassView.java
├─ myclass.css
├─ myclass.properties
└─ myclass.fxml
```

Es werden eine `*View.java` und eine `*Presenter.java` Klasse erwartet, sowie eine Cascading Style Sheets (CSS)-Datei, für die Gestaltung der View, eine `*.properties`-Datei mit Key-Value Inhalt, z. B. für Strings, und die FXML-Datei für das GUI-Layout erwartet. Die `.css` und `.properties` Dateien benötigen nicht zwingend einen Inhalt, sollten jedoch vorhanden sein.

Neben der vereinfachten Oberflächenerstellung bietet das Framework noch eine zweite nützliche Eigenschaft. Es erlaubt, ohne weitere Konfiguration, Java EE Annotationen (`javax.inject.*`) für Dependency Injection zu benutzen, wozu andernfalls Frameworks wie *Google Guice*<sup>8</sup>, benötigt würden, die eine umfangreiche Konfiguration verlangen. [Vgl. Bie14]

Die Verbindung zum Server wird auch hier über einen REST-Client hergestellt.

### 5.1.4 REST-Client

Als REST-Client wird der Client der Jersey 2 Implementierung verwendet. Für Aufrufe erfolgen fließend und erlauben es unnötige Zuweisungen zu vermeiden, z. B.

---

<sup>8</sup>Google Guice Webseite: <https://code.google.com/p/google-guice/>



```
Response r = client.target(getBaseUri()).path(<Pfad>).request().head();
```

Aus dem Response Objekt können Objekte mit `.readEntity()` abgerufen werden, dazu muss der Typ des Abzurufenden Objekts im Aufruf übergeben werden und die Entitäten werden automatisch deserialisiert. [Vgl. Bur14, S. 113 ff.]

## 5.2 Weiterentwicklung der Android Bibliothek

Es wurden die folgenden Änderungen an der Android Bibliothek durchgeführt:

- Anpassen des REST-Clients an den neuen Server.
- Verwendung geteilter Entitätsklassen zwischen Bibliothek und Server.
- Überarbeitung der Bildschirmfassung und Visualisierung von Eingaben.
- Anpassung der Oberfläche an die veränderte Terminologie und kleinere Verbesserungen.

Die Screenshots im Anhang C.0.3 zeigen einen Teil der Änderungen. MotionEvents werden nun direkt an die Bibliothek weitergereicht. Dies führt dazu, dass alle Eingaben registriert werden, auch in Views die diese normalerweise konsumieren und so nicht mehr weitergeben.

## 5.3 Testen der Software

Die Oberfläche wurde manuell getestet und für den REST-Client wurde JUnit verwendet. Die Android Anwendung wurde ausschließlich manuell getestet.

## 6 Ergebnisse

### 6.1 Server

Die Arbeit an der Serverkomponente war die umfangreichste am Projekt. Probleme mit der *Glassfish* Konfiguration und dessen teilweise unzureichende Dokumentation haben sehr viel Zeit in Anspruch genommen. Das Ergebnis ist ein Server mit Datenbankanbindung und REST-API, die weitgehend implementiert wurde. Einzig Funktionen zum Dateiupload konnten aus Zeitgründen nicht mehr implementiert werden, wodurch die Android Bibliothek die gesammelten Daten nicht hochladen kann und die Daten entsprechend nicht für die Auswertung bereitstehen.

### 6.2 Testleiter-Client und Android Bibliothek

Umgesetzt wurden die Anmeldung und die Benutzerverwaltung. Die Kommunikation mit dem Server wurde in Ansätzen implementiert, verläuft aktuell jedoch synchron im UI-Thread, was zum Blockieren der GUI führt, wenn große Datenmengen abgerufen werden. Um dies zu verhindern sollte der Datentransfer in JavaFX Tasks nebenläufig durchgeführt werden, dies konnte jedoch in der vorgegebenen maximalen Bearbeitungsdauer nicht mehr umgesetzt werden.

Die Android Bibliothek wurde ein wenig verbessert, bei der Erkennung von Eingaben und Gesten und die Persistenzschicht wurde eingeführt, ist jedoch noch nicht voll implementiert. Es wurde viel Zeit investiert, um eine bessere Möglichkeit zu finden den Bildschirminhalt zu erfassen, ohne Root-Zugriff das System zu haben. Dies verlief jedoch weitgehend ergebnislos. Zwar wäre es möglich eine Systemapplikation zu erstellen, dies würde allerdings die Integration in andere Anwendungen erschweren, da Systemanwendungen eine besondere Signatur erfordern.

## 7 Fazit

Die Durchführung des Projekts führte zu vertieftem Wissen über verschiedene aktuelle Technologien. Dazu gehören ein detaillierter Einblick in Java EE und JavaFX, sowie Konzepte wie z.B. Dependency Injection. Das ursprüngliche Ziel war die Entwicklung eines funktionsfähigen Frameworks, welches aus zwei neu implementierten und einer angepassten Komponente besteht. Aufgrund der fehlenden Vertrautheit mit den verwendeten Technologien (JavaFX, Java EE 7) und den teilweise schwer ergründbaren Fehlern, die bei der Entwicklung des Servers auftraten, war der Projektumfang deutlich zu groß angesetzt, für eine alleinige Bearbeitung. Diese Fehleinschätzung führte dazu, dass das Projektziel nicht in vollem Umfang erreicht wurde.

Jedoch wurde eine brauchbare Grundlage mit verschiedenen Beispielen der hier behandelten Konzepte und Technologien geschaffen, auf der sich aufbauen lässt. Um dieses Framework fertigzustellen, inklusive Unit-Tests und erweiterter Testanalyse müssten jedoch noch viel Zeit investiert werden, was im Rahmen der Projektbearbeitung nicht möglich wäre. Da das Thema weiterhin spannend ist und Potential enthält wird die Bearbeitung außerhalb des Rahmens des Projekts weitergeführt. Im nächsten Abschnitt wird kurz auf mögliche Richtungen zur Weiterentwicklung eingegangen.

### 7.1 Weiterentwicklung

#### 7.1.1 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.

### 7.1.2 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 Tablet-PCs. 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 Bildschirm-inhalts 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.

## 8 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).
- [And14a] Android Developers, Hrsg. *Android Debug Bridge*. 2014. URL: <http://developer.android.com/tools/help/adb.html> (besucht am 05.10.2014).
- [And14b] Android Developers, Hrsg. *Android KitKat*. 2014. URL: <https://developer.android.com/about/versions/kitkat.html> (besucht am 05.10.2014).
- [And14c] Android Developers, Hrsg. *Get the Android SDK*. 2014. URL: <http://developer.android.com/sdk/index.html> (besucht am 02.10.2014).
- [Bie14] Adam Bien. *afterburner.fx*. 2014. URL: <http://afterburner.adam-bien.com/> (besucht am 06.10.2014).
- [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/javase/javase7-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).
- [Dea14] Carl Dea. *JavaFX 8: Introduction by example*. Second Edition. 2014. ISBN: 1430264608.
- [EH14] Oliver Erxleben und Albert Hoffmann. *Entwicklung eines Usability-Test-Frameworks für Android*. 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).

- [Goo14a] Google, Hrsg. *Chromecast*. 2014. URL: <https://www.google.de/chrome/devices/chromecast/> (besucht am 05. 10. 2014).
- [Goo14b] Google Inc. *Chromecast*. 2014. URL: <https://play.google.com/store/apps/details?id=com.google.android.apps.chromecast.app> (besucht am 05. 10. 2014).
- [Joy14] Joyent, Inc, Hrsg. *About Node.js@#*. 2014. URL: <http://nodejs.org/about/> (besucht am 02. 10. 2014).
- [Jun13] Josh Juneau. *Java EE 7 recipes: A problem-solution approach*. The expert's voice in Java. Berkeley, Calif. und New York: Apress; Distributed to the book trade worldwide by Springer, 2013. ISBN: 9781430244257.
- [KS13] Mike Keith und Merrick Schincariol. *Pro JPA 2*. Second Edition. The expert's voice in Java. 2013. ISBN: 9781430249269.
- [Ora14a] Oracle, Hrsg. *Java EE 7 Technologies*. 2014. URL: <http://www.oracle.com/technetwork/java/javaee/tech/index.html> (besucht am 02. 10. 2014).
- [Ora14b] Oracle. *MySQL Workbench*. 2014. URL: <http://dev.mysql.com/downloads/workbench/>.
- [Ora13a] Oracle, Hrsg. *The Java EE 6 Tutorial: A Singleton Session Bean Example: counter*. 2013. URL: <http://docs.oracle.com/javaee/6/tutorial/doc/gipvi.html> (besucht am 05. 10. 2014).
- [Ora14c] Oracle Corporation, Hrsg. *Chapter 13 Using Connector/J with GlassFish*. 2014. URL: <http://dev.mysql.com/doc/connector-j/en/connector-j-usagenotes-glassfish-config.html> (besucht am 05. 10. 2014).
- [Ora13b] 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).
- [Ora14d] Oracle Corporation. *JavaFX Scene Builder*. 2014. URL: <http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html> (besucht am 06. 10. 2014).
- [Ora14e] Oracle Corporation, Hrsg. *MySQL.com*. 2014. URL: <http://www.mysql.com/> (besucht am 05. 10. 2014).
- [Ran13] Random Stuff, Hrsg. *Grab Android screenshot to computer via ADB*. 2013. URL: <http://blog.shvetsov.com/2013/02/grab-android-screenshot-to-computer-via.html> (besucht am 05. 10. 2014).
- [Sha11] Remy Sharp. *Two videos in sync*. Hrsg. von HTML5 demos. 2011. URL: <http://html5demos.com/two-videos> (besucht am 03. 10. 2014).

- [Sou10] Kyle Sourcy. *Unmoderated, Remote Usability Testing: Good or Evil?* Hrsg. von UXmatters. 2010. URL: <http://www.uxmatters.com/mt/archives/2010/01/unmoderated-remote-usability-testing-good-or-evil.php> (besucht am 06. 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

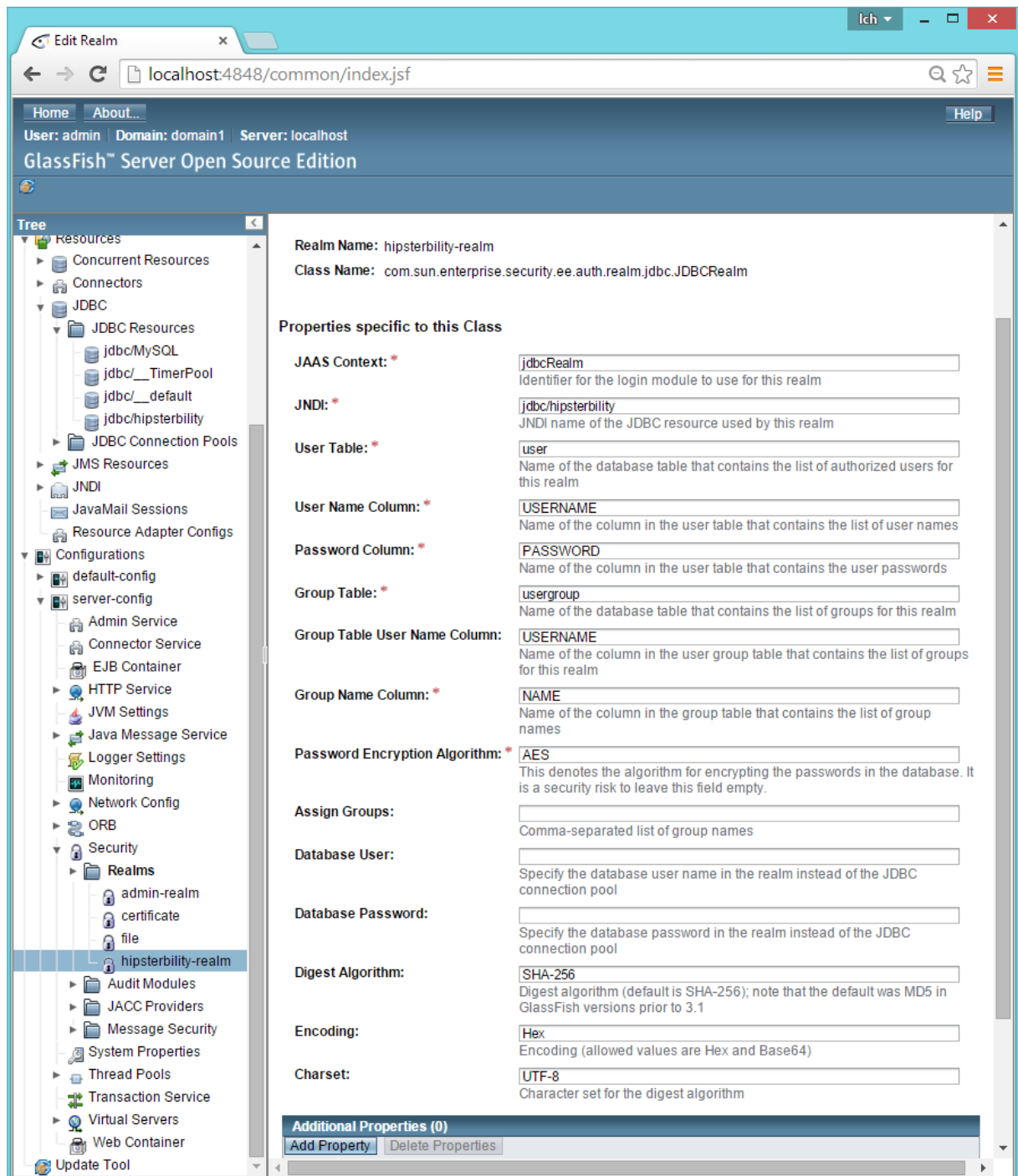


Abbildung 4: Screenshot der Security-Realm Konfiguration des Glassfish Servers



## A.2 Glassfish Befehle

```

1 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 7: Erstellen eines Glassfish Security Realms mit dem „asadmin“ Werkzeug.

## A.3 Konfigurationsdateien

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish
  Application Server 3.1
3   Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-
  resources_1_5.dtd">
4
5 <resources>
6   <jdbc-connection-pool name="jdbc_hipsterbilityPool"
7     res-type="javax.sql.
8       ConnectionPoolDataSource"
9       datasource-classname="com.mysql.jdbc.jdbc2.
10         optional.MySQLDataSource">
11
12     <property name="User" value="hipsterbility" />
13     <property name="Port" value="3306" />
14     <property name="DatabaseName" value="hipsterbility" />
15     <property name="ServerName" value="localhost" />
16     <property name="Url" value="jdbc:mysql://localhost:3306/
17       hipsterbility"/>
18     <property name="URL" value="jdbc:mysql://localhost:3306/
19       hipsterbility"/>
20     <property name="Password" value="hipsterbility123" />
21   </jdbc-connection-pool>
22
23   <jdbc-resource enabled="true"
24     jndi-name="jdbc/hipsterbility"
25     object-type="user"
26     pool-name="jdbc_hipsterbilityPool">
27     <description />
28   </jdbc-resource>
29 </resources>

```

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

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish
  Application Server 3.1 Servlet 3.0//EN" "http://glassfish.org/dtds
  /glassfish-web-app_3_0-1.dtd">

```

```

3 <glassfish-web-app error-url="">
4     <context-root>hipsterbility</context-root>
5     <security-role-mapping>
6         <role-name>ADMIN</role-name>
7         <group-name>ADMIN</group-name>
8     </security-role-mapping>
9     <security-role-mapping>
10        <role-name>USER</role-name>
11        <group-name>USER</group-name>
12    </security-role-mapping>
13    <class-loader delegate="true"/>
14    <jsp-config>
15        <property name="keepgenerated" value="true">
16            <description>Keep a copy of the generated servlet class'
17                java code.</description>
18        </property>
19    </jsp-config>
20    <keep-state>>false</keep-state>
21 </glassfish-web-app>

```

Listing 9: glassfish-web.xml JSP Konfiguration und Mapping von SecurityRealm Rollen.

## A.4 Scriptdateien

### A.4.1 Befehlsdateien

```

1 add-resources glassfish-resources.xml
2 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 10: glassfish-create.txt Befehle zu erzeugen von Ressourcen.

```

1 delete-auth-realm hipsterbility-realm
2 delete-jdbc-resource jdbc/hipsterbility
3 delete-jdbc-connection-pool jdbc_hipsterbilityPool

```

Listing 11: glassfish-delete.txt Befehle zum Löschen von Ressourcen.

## Windows

```

1 asadmin multimode --file glassfish-create.txt

```

Listing 12: glassfish-create-resources.bat

```
1 asadmin multimode --file glassfish-delete.txt
```

Listing 13: glassfish-delete-resources.bat

## Unix / Linux

```
1 #!/bin/bash
2 # add path to glassfish installation if necessary
3 asadmin multimode --file glassfish-create.txt
```

Listing 14: glassfish-create-resources.sh

```
1 #!/bin/bash
2 # add path to glassfish installation if necessary
3 asadmin multimode --file glassfish-delete.txt
```

Listing 15: glassfish-delete-resources.sh

## B Serverkonfiguration

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="
  "2.1">
3   <persistence-unit name="HipsterbilityService" transaction-type="
    RESOURCE_LOCAL">
4     <jta-data-source>jdbc/hipsterbility</jta-data-source>
5     <exclude-unlisted-classes>false</exclude-unlisted-classes>
6     <properties>
7       <property name="eclipselink.logging.level" value="FINE"/>
8       <property name="eclipselink.ddl-generation.output-mode"
9         value="both"/>
10      <property name="eclipselink.ddl-generation"
11        value="create-or-extend-tables"/>
12    </properties>
13  </persistence-unit>
14 </persistence>
```

Listing 16: persistence.xml (Datenbankverbindung)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5     http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
6   version="3.0">
7   <context-param>
8     <param-name>configuration</param-name>
9     <param-value>development</param-value>
10  </context-param>
11
```

```

12     <servlet>
13         <servlet-name>Jersey Web Application</servlet-name>
14         <servlet-class>org.glassfish.jersey.servlet.ServletContainer<
            /servlet-class>
15         <init-param>
16             <param-name>javax.ws.rs.Application</param-name>
17             <param-value>de.hsosnabrueck.hipsterbility.config.
                ApplicationConfig</param-value>
18         </init-param>
19         <load-on-startup>1</load-on-startup>
20     </servlet>
21     <servlet>
22         <servlet-name>Faces Servlet</servlet-name>
23         <servlet-class>javax.faces.webapp.FacesServlet</servlet-class
            >
24         <load-on-startup>1</load-on-startup>
25     </servlet>
26     <servlet-mapping>
27         <servlet-name>Jersey Web Application</servlet-name>
28         <url-pattern>/api/*</url-pattern>
29     </servlet-mapping>
30     <servlet-mapping>
31         <servlet-name>Faces Servlet</servlet-name>
32         <url-pattern>*.xhtml</url-pattern>
33     </servlet-mapping>
34     <welcome-file-list>
35         <welcome-file>/index.jsp</welcome-file>
36     </welcome-file-list>
37
38     <security-constraint>
39         <web-resource-collection>
40             <web-resource-name>Rest API</web-resource-name>
41             <url-pattern>/api/*</url-pattern>
42         </web-resource-collection>
43         <!--<auth-constraint>-->
44             <!--<description>constraint for REST API</description>-->
45             <!--<role-name>ADMIN</role-name>-->
46             <!--<role-name>USER</role-name>-->
47         <!--</auth-constraint>-->
48     </security-constraint>
49     <security-constraint>
50         <web-resource-collection>
51             <web-resource-name>User Resources</web-resource-name>
52             <url-pattern>/api/users/*</url-pattern>
53             <http-method-omission>POST</http-method-omission><!--
                allow POST for all users-->
54         </web-resource-collection>
55         <auth-constraint>
56             <role-name>ADMIN</role-name>
57             <role-name>USER</role-name>
58         </auth-constraint>
59     </security-constraint>
60     <security-constraint>
61         <web-resource-collection>
62             <web-resource-name>Sessions Resources</web-resource-name>

```

```

63         <url-pattern>/api/tests/*</url-pattern>
64     </web-resource-collection>
65     <auth-constraint>
66         <role-name>ADMIN</role-name>
67         <role-name>USER</role-name>
68     </auth-constraint>
69 </security-constraint>
70 <security-constraint>
71     <web-resource-collection>
72         <web-resource-name>Tests Resources</web-resource-name>
73         <url-pattern>/api/tests/*</url-pattern>
74     </web-resource-collection>
75     <auth-constraint>
76         <role-name>ADMIN</role-name>
77         <role-name>USER</role-name>
78     </auth-constraint>
79 </security-constraint>
80 <security-constraint>
81     <web-resource-collection>
82         <web-resource-name>Devices Resources</web-resource-name>
83         <url-pattern>/api/devices/*</url-pattern>
84     </web-resource-collection>
85     <auth-constraint>
86         <role-name>ADMIN</role-name>
87         <role-name>USER</role-name>
88     </auth-constraint>
89 </security-constraint>
90 <security-constraint>
91     <web-resource-collection>
92         <web-resource-name>Devices Resources</web-resource-name>
93         <url-pattern>/api/invites/*</url-pattern>
94     </web-resource-collection>
95     <auth-constraint>
96         <role-name>ADMIN</role-name>
97     </auth-constraint>
98 </security-constraint>
99 <login-config>
100     <auth-method>BASIC</auth-method>
101     <realm-name>hipsterbility-realm</realm-name>
102 </login-config>
103 <security-role>
104     <description>Standard service users</description>
105     <role-name>USER</role-name>
106 </security-role>
107 <security-role>
108     <description>Admin and test supervisors</description>
109     <role-name>ADMIN</role-name>
110 </security-role>
111 </web-app>

```

Listing 17: web.xml (Servlet Konfiguration)

## B.1 Diagramme

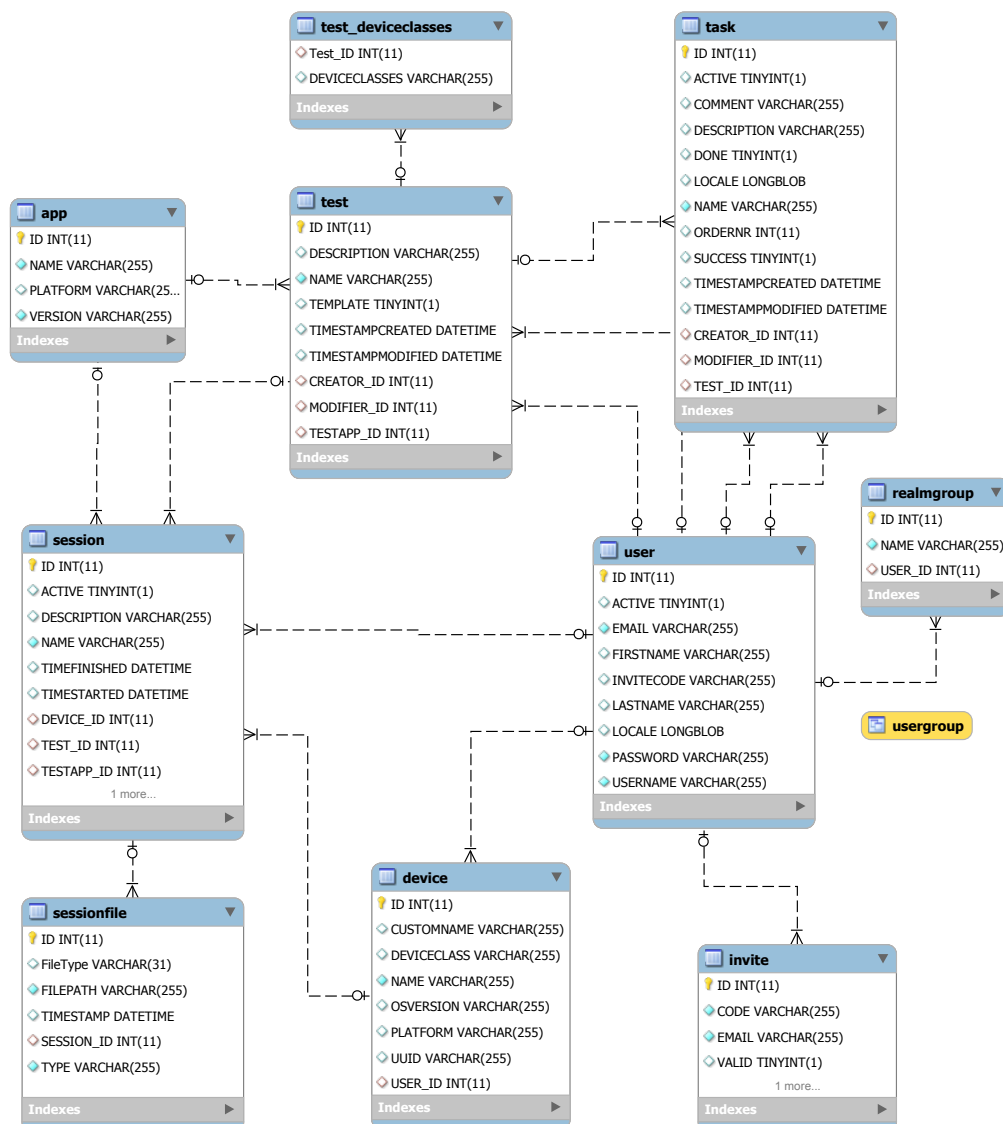


Abbildung 5: Datenbankschema des Hipsterbility-Servers.

## C Screenshots

### C.0.1 JavaFX Scene Builder

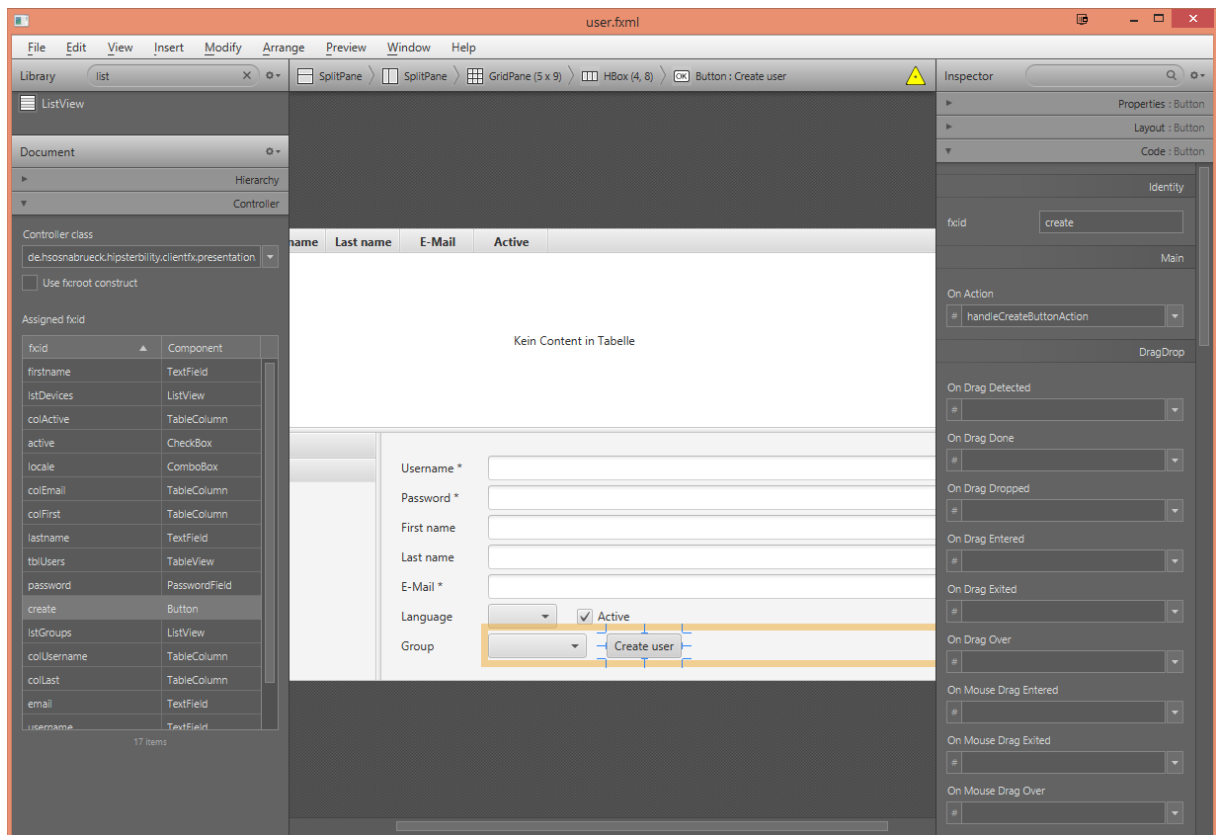
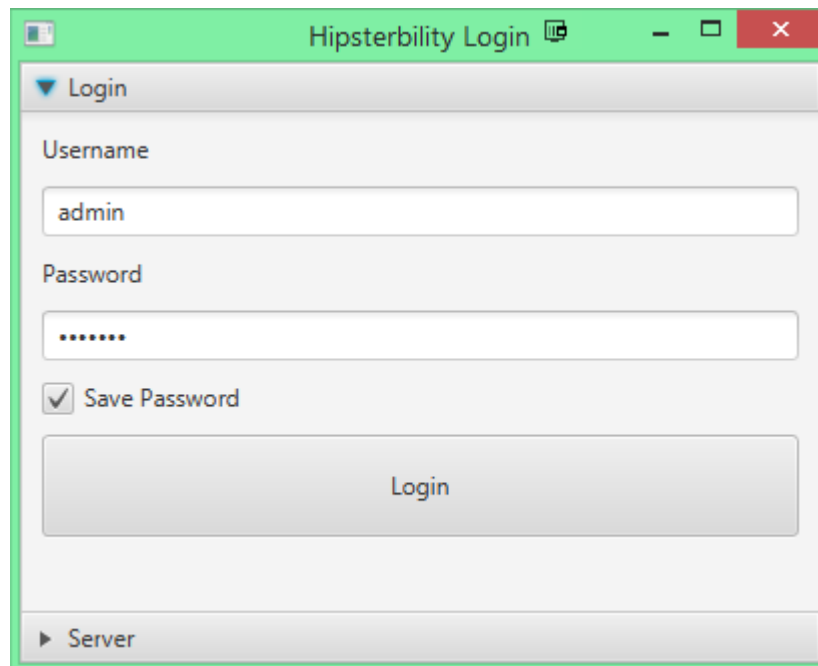


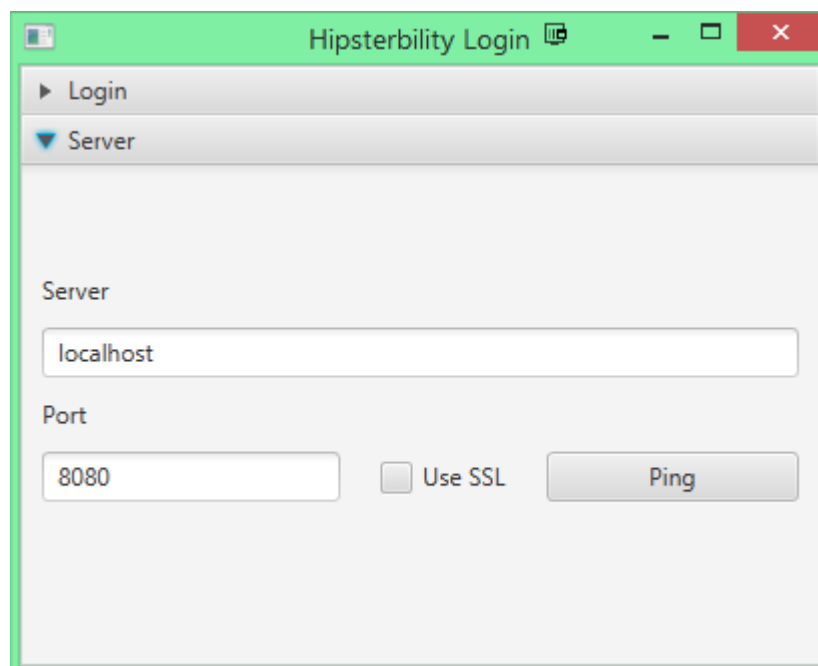
Abbildung 6: Screenshot des JavaFX Scene Builders mit geöffneter FXML Oberfläche.

### C.0.2 Testleiter Client



The screenshot shows a window titled "Hipsterbility Login". It has a tab labeled "Login" which is currently selected. Below the tab, there are two text input fields: "Username" containing the text "admin" and "Password" containing seven dots. Below these fields is a checked checkbox labeled "Save Password". At the bottom of the form is a large "Login" button. At the very bottom of the window, there is a tab labeled "Server" which is not selected.

Abbildung 7: Client Login Fenster.



The screenshot shows the same "Hipsterbility Login" window, but the "Server" tab is now selected. The "Login" tab is collapsed. The "Server" tab contains a "Server" text input field with "localhost" entered, a "Port" text input field with "8080" entered, a "Use SSL" checkbox which is unchecked, and a "Ping" button.

Abbildung 8: Client Login Fenster, Server Einstellungen.



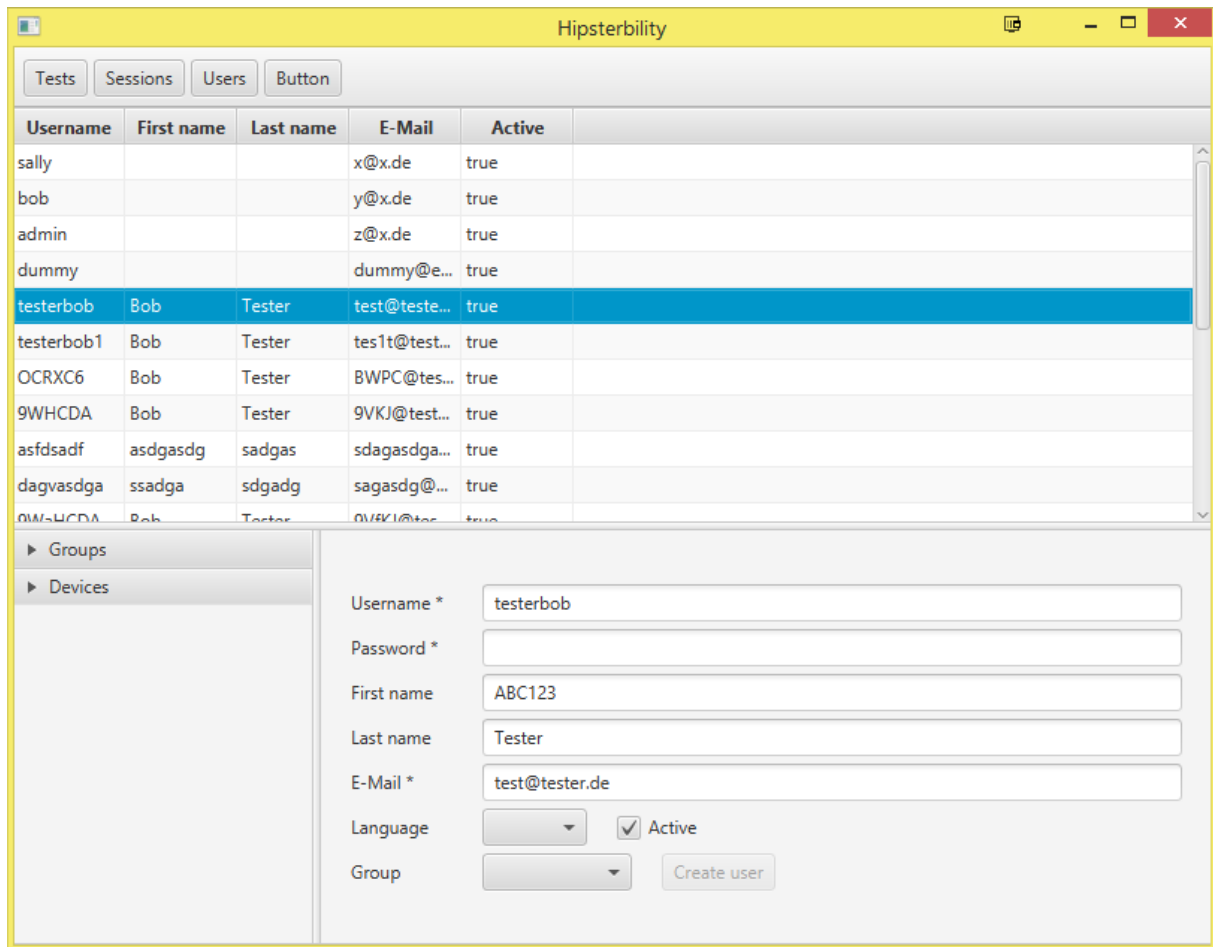


Abbildung 9: Client Benutzerverwaltung mit Testdaten.

### C.0.3 Android Testapplikation mit Bibliothek

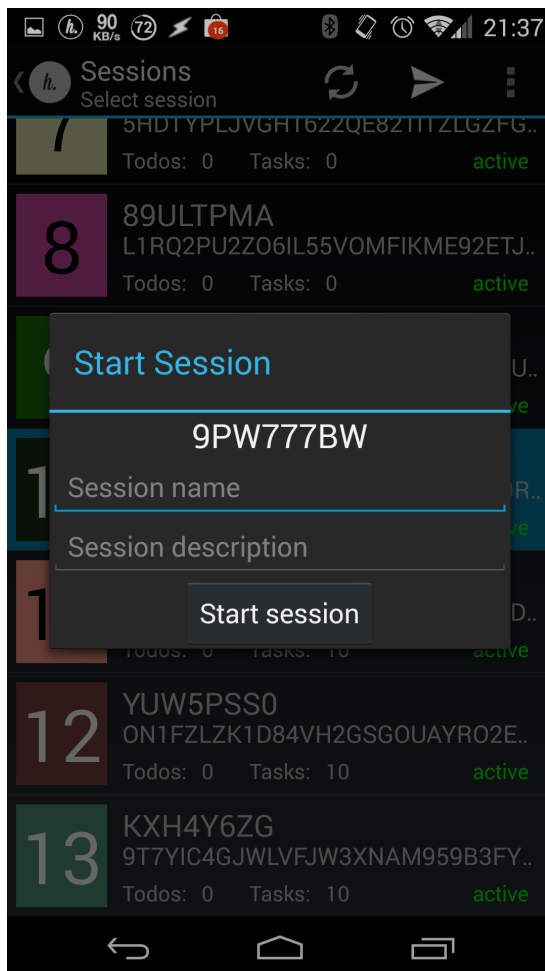


Abbildung 10: Android Bibliothek, Dialog zum Starten einer Testsitzung.

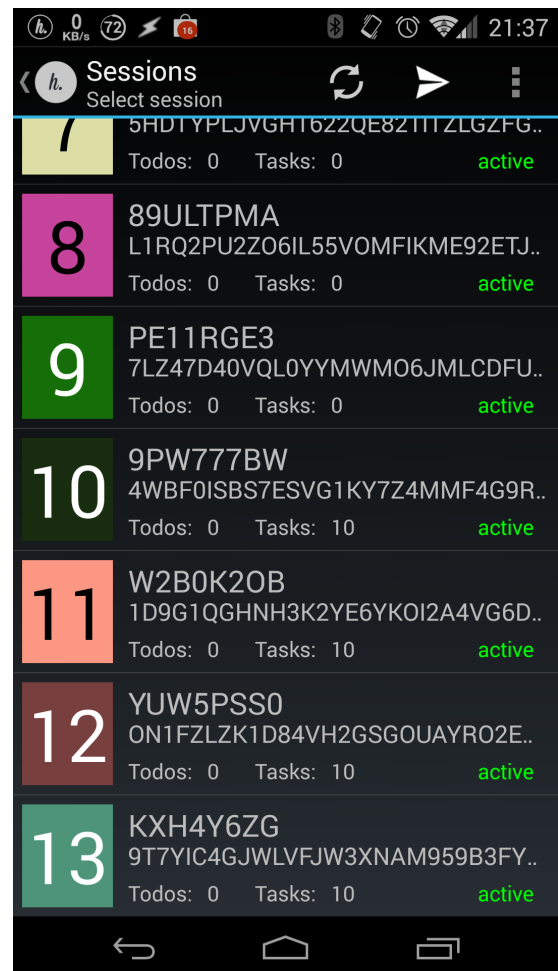


Abbildung 11: Android Bibliothek, Liste mit Test.



Abbildung 12: Android Bibliothek, Liste mit Aufgaben zu einem Test.

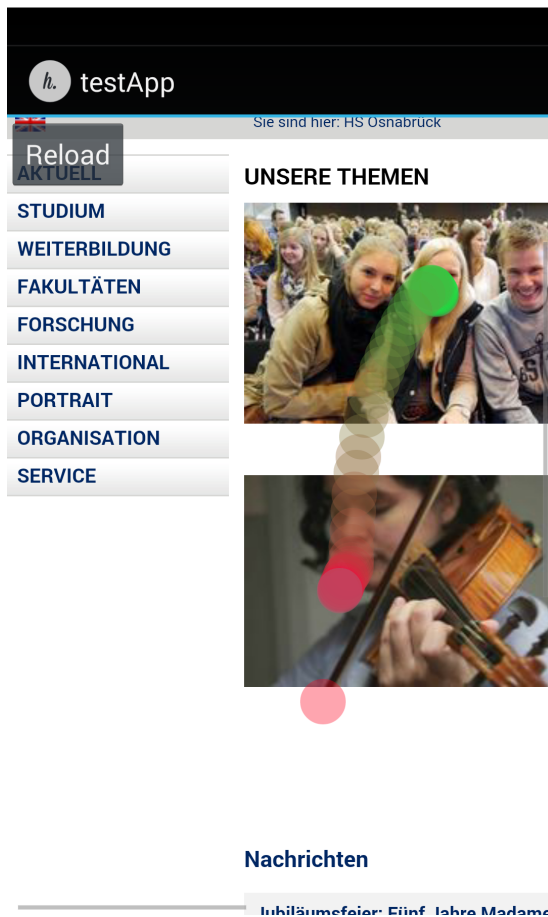


Abbildung 13: Android Bibliothek, erfasste Geste in einer WebView.

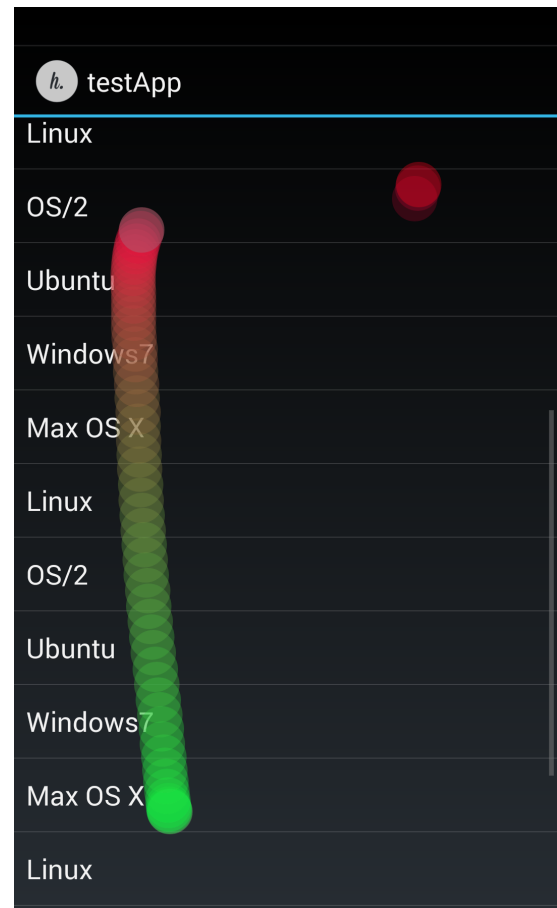


Abbildung 14: Android Bibliothek, bei der Aufzeichnung erfasste Geste und zweiter Berührungspunkt.

# 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)