

Universität Hamburg
Fachbereich Informatik

Bachelorarbeit

**Realisierung eines Single-Sign-On-Dienstes
für das Informatik-Netz**

vorgelegt von

Senad Ličina

geb. am 23.03.1988 in Berane (MNE)

Matrikelnummer 5945457

Studiengang Informatik

eingereicht am 20. Mai 2013

Betreuer: Dipl.-Wirtsch.-Inf. Dominik Herrmann

Erstgutachter: Prof. Dr.-Ing. Hannes Federrath

Zweitgutachter: Dr. Andreas Günter

Aufgabenstellung

Im Informatik-Rechenzentrum der Universität Hamburg werden die Benutzerdaten aller Mitarbeiter und Studenten in einem LDAP-Verzeichnis (MS ActiveDirectory) vorgehalten.

Im Rahmen einer Bachelorarbeit ist auf Basis dieser Infrastruktur ein webbasierter Single-Sign-On-Dienst (SSO-Dienst) zu implementieren, der von verschiedenen (Web-)Anwendungen genutzt werden kann. Dies erlaubt es den Entwicklern von Web-Anwendungen, den sensiblen Prozess der Benutzerauthentifizierung, bei der das vom Benutzer eingegebene Passwort übermittelt und geprüft werden muss, an den SSO-Dienst auszulagern.

Die Nutzung des SSO-Dienstes soll möglichst über (Standard-)Schnittstellen, etwa OAuth 2.0, möglich sein. Im Rahmen der Bachelorarbeit ist ein Entwurf für den Dienst anzufertigen und ein Prototyp zu implementieren. Weiterhin sind Beispiel-Implementierungen sowie eine leicht verständliche Dokumentation mit Tutorial-Charakter für die wesentlichen Entwicklungsumgebungen (z.B. PHP, Java, Ruby on Rails) anzufertigen. Die Umsetzung muss hohen Sicherheitsanforderungen genügen.

Zusammenfassung

Authentifizierungs- und Autorisierungsdienste gewinnen im World Wide Web zunehmend an Bedeutung. Sie erleichtern nicht nur die Entwicklung von Web-Anwendungen sondern bieten Benutzern eine Möglichkeit sich einfach, einheitlich und sicher zu Authentifizieren und Anwendungen mit dem Zugriff auf ihre Ressourcen zu Autorisieren. Der Fachbereich Informatik der Universität Hamburg stellt bisher keinen solchen Dienst zur Verfügung.

Diese Bachelorarbeit beschäftigt sich mit der Implementierung von *OAuth InfRZ*, eines im Zuge dieser Arbeit programmierten Single-Sign-On-Dienstes mit Zugriff auf die Benutzer-Accounts des Informatik Rechenzentrums. Sie richtet sich primär an Entwickler von Client-Anwendungen¹ und Entwickler von *OAuth InfRZ*. Diese Arbeit bildet eine technische Dokumentation, begründet Design-Entscheidungen im Prozess der Implementierung und beschäftigt sich mit Kernthemen des Aufgabengebiets.

OAuth InfRZ basiert auf dem *OAuth*-Protokoll und greift mit dem *Lightweight Directory Access Protocol* auf die im *Active Directory* des Informatik Rechenzentrums verwaltete Benutzer-Information zu. Benutzern wird es ermöglicht sich gegenüber dritten Anwendungen zu authentifizieren und diese mit dem Zugriff auf ihre Ressourcen zu autorisieren. Entwicklern von Client-Anwendungen wird eine leicht verständliche *Client-Library* geboten, welche die Benutzung von *OAuth InfRZ* komfortabel gestaltet. Durch den Einsatz von *OAuth InfRZ* wird sowohl die Benutzung von Fachbereichsinternen Diensten als auch die Entwicklung solcher Dienste vereinheitlicht und vereinfacht.

¹Anwendung die einen *OAuth*-Dienst zu Authentifizierung der Benutzer und/oder den Zugriff auf deren Information benutzen

Inhaltsverzeichnis

1. Einleitung	1
2. Grundlagen	3
2.1. Authentifizierung und Autorisierung	3
2.2. Transport Layer Security	3
2.2.1. Funktionsweise & Verschlüsselung	4
2.2.2. Sicherheit	4
2.3. Active Directory	4
3. OAuth	6
3.1. Authentifizierung im World Wide Web	6
3.2. Versionsgeschichte	7
3.3. OAuth 2 Workflow	7
3.4. Sicherheit	9
4. OAuth InfRZ	10
4.1. Anforderungen	10
4.2. Benutzung	11
4.2.1. Benutzerrollen	11
4.2.2. Menüleiste	12
4.2.3. Hauptseite	12
4.2.4. Authentifizierung	13
4.2.5. Autorisierung	13
4.2.6. Client-Management	14
4.3. Eingesetzte Werkzeuge	18
4.3.1. Back-End: PHP 5.4, SQLite, nginx	18
4.3.2. Front-End: HTML5, CSS3, JavaScript	19
4.3.3. Frameworks: Twig, Bootstrap, jQuery	20
4.3.4. Verschlüsselung: TLS	20
4.4. Eingesetzte Konzepte	21
4.5. Technische Realisierung	22
4.5.1. Versionierung	22
4.5.2. Verzeichnisstruktur und Dateien	22
4.5.3. Front-Controller	23
4.5.4. Auth-Factory	24
4.5.5. Sicherheit	25
4.6. Verfügbare Request-Endpunkte	28
4.7. Client-Library	29
4.7.1. Benutzung	29
4.7.2. Konfiguration	30
4.7.3. Client-Server Kommunikation	31

4.8. Ausblick	33
4.8.1. Client-Library	33
4.8.2. Host	33
4.8.3. Client-Moderator	34
Literaturverzeichnis	35
Anhang	36
Selbständigkeitserklärung	37

1. Einleitung

Ein **Single-Sign-On-Dienst (SSO-Dienst)** bietet dem Benutzer die Möglichkeit nach einer einmaligen Authentifizierung auf mehrere Ressourcen und Dienste zuzugreifen. Der Einsatz eines SSO-Dienstes soll die Benutzung komfortabler und sicherer gestalten. In [Hur97] wird Single-Sign-On definiert, damit verbundene Implementierungsschwierigkeiten erörtert und zwischen unterschiedlichen Modellen unterschieden.

Im Zeitalter des Web 2.0² wird ein simpler und einheitlicher Autorisierungsmechanismus immer wichtiger. Besonders in den letzten Jahren hat sich das offene Protokoll **OAuth** zu diesem Zweck durchgesetzt. Obwohl OAuth primär der Autorisierung dient, bieten die meisten großen Web-Portale OAuth zur Authentifizierung der Benutzer und Autorisierung dritter Anwendungen an. Der Einsatz eines SSO-Dienstes erleichtert die Entwicklung weiterer unabhängiger Dienste und kann deren Sicherheit durch die Übernahme sicherheitsrelevanter Kernfunktionen erhöhen. Neben hohen Sicherheitsanforderungen, denen ein solcher Dienst entsprechen muss, sollte er auch vertrauenswürdig, benutzerfreundlich, selbsterklärend, wart- und erweiterbar sein.

Das Informatik Rechenzentrum der Universität Hamburg benutzt zur Verwaltung und Bereitstellung der Benutzerdaten den Verzeichnisdienst **Active Directory**. Durch das Ansprechen des Active Directory wird es Benutzern ermöglicht sich mit ihren Informatik-Rechenzentrum-Anmeldedaten (Credentials)³ zu authentifizieren. Diese Authentifizierung kann gegen alle beim OAuth-Dienst angemeldeten Dienste erfolgen. Desweiteren können Benutzer angemeldete Dienste für den Zugriff auf ihre Informationen autorisieren. Der Einsatz eines OAuth-Dienstes entkoppelt die Authentifizierung von der Funktionalität weiterer Dienste und kann so deren Sicherheit erhöhen. Eine Reduzierung der erforderlichen Authentifizierungen⁴ wirkt sich ebenfalls positiv auf die Sicherheit aus. Ein selbsterklärender, transparenter und einheitlicher Authentifizierungs- und Autorisierungsvorgang kann zu mehr Vertrauen bei den Benutzern führen.

Bei diesem Dokument handelt es sich um einen Teil der Dokumentation zu OAuth InfrZ, ein vom Autor im Frühjahr 2013 entwickelter SSO-Dienst für den Fachbereich Informatik der Universität Hamburg. OAuth InfrZ ist Bestandteil dieser Bachelorarbeit.

Dieses Dokument ist wie folgt aufgebaut:

Kapitel 2 bringt den Leser auf den zum Verständnis der Arbeit erforderlichen Wissensstand, indem es wichtige **Grundlagen** und die damit verbundenen Fragestellungen klärt. Es wird auf Authentifizierung, Autorisierung, ein Verschlüsselungsprotokoll für

²vgl. Tim O'Reilly: What Is Web 2.0. <http://oreilly.com/web2/archive/what-is-web-20.html> (Zugriff am 26.02.2013)

³die Benutzerkennung und das dazugehörige Passwort

⁴und damit die Anzahl der Übertragungen der sensiblen Credentials

Netzwerkübertragungen und den vom Fachbereich Informatik genutzten Verzeichnisdienst eingegangen.

In Kapitel 3 wird **OAuth** erklärt und seine Funktionsweise beschrieben. Es werden die Vorteile von OAuth gegenüber herkömmlicher Authentifizierung erörtert, die Versionen gegenübergestellt, der OAuth2 Workflow skizziert und die Sicherheit bewertet.

Kapitel 4 enthält eine technische Dokumentation zu **OAuth InfRZ**. Zunächst werden die Anforderungen an ein solches System skizziert und die Benutzung von OAuth InfRZ erklärt. Um die Einarbeitungszeit von weiteren Entwicklern zu reduzieren, werden die eingesetzten Werkzeuge und Konzepte vorgestellt und komplexe Besonderheiten zur technischen Realisierung ausgeführt. Außerdem wird die Client-Library vorgestellt und ein Ausblick mit Aufwandsbewertung auf die Erweiterbarkeit von OAuth InfRZ geboten.

2. Grundlagen

In diesem Kapitel werden für das Verständnis dieses Dokuments notwendige Konzepte und Technologien diskutiert. Zunächst werden Authentifizierung und Autorisierung unterschieden. Dann wird das Verschlüsselungsprotokoll Transport Layer Security (TLS) erklärt, seine Funktionsweise skizziert und seine Sicherheit erörtert. Außerdem werden der Verzeichnisdienst Active Directory (AD) und das Lightweight Directory Access Protocol (LDAP) erklärt und in einen Zusammenhang gebracht.

2.1. Authentifizierung und Autorisierung

Als **Authentifizierung** wird die Verifizierung behaupteter Eigenschaften bezeichnet.⁵ Im speziellen ist bei einer Authentifizierung im World Wide Web (WWW) meist die Prüfung und Erteilung einer Zugangsberechtigung gemeint. Dies kann durch den Login eines Benutzers bei einer Webanwendung geschehen. Der Benutzer authentisiert sich bei der Webanwendung indem er sein Passwort übermittelt. Im Gegenzug authentifiziert die Webanwendung den Benutzer.

Die **Autorisierung** ist das Einräumen von Zugriffsrechten für einen Benutzer oder ein Programm. Eine Autorisierung setzt eine erfolgreichen Authentifizierung voraus.

„As an aside, some people confuse authorization with authentication. Authentication deals with the question of whether you are actually communicating with a specific process. Authorization is concerned with what that process is permitted to do.”

– [TW10] Section 8.7

2.2. Transport Layer Security

Bei **Transport Layer Security (TLS)**⁶ handelt es sich um ein *hybrides Verschlüsselungsprotokoll*.⁷ Es wird von der Internet Engineering Task Force (IETF) gepflegt, welche im August 2008 mit [RFC5246] die aktuellste Version *TLS 1.2* vorstellte.

Im Transmission Control Protocol/Internet Protocol Reference Model (TCP/IP-Modell)⁸ wird TLS zwischen der Transport- und Anwendungsschicht angesiedelt. [TW10, Section 8.9.3] gliedert es in eine eigene „*Security*“ Schicht. Im Open Systems Interconnection Reference Model (OSI-Modell)⁹ wird es in die Kommunikationsschicht gegliedert.¹⁰ Es wird hauptsächlich zur Sicherung von HTTP-Verbindungen eingesetzt.

⁵vgl. [TW10, Section 8.7]

⁶auch bekannt unter der Vorgängerbezeichnung *Secure Sockets Layer (SSL)*

⁷vgl. [WK06, Hannes Federrath, Andreas Pfitzmann: „IT-Sicherheit“. Kapitel 2.4 (S. 279-281)]

⁸vgl. [TW10, Section 1.4.2]

⁹vgl. [TW10, Section 1.4.1]

¹⁰vgl. [Sin12, Section 12.4]

2.2.1. Funktionsweise & Verschlüsselung

Als ein **hybrides Verschlüsselungsprotokoll** setzt TLS sowohl auf *symmetrische* als auch auf *asymmetrische* Verschlüsselung. Im ersten Schritt handeln die Kommunikationspartner die Sicherheitsparameter durch asymmetrische Verschlüsselungsverfahren aus und haben die Möglichkeit sich durch Zertifikate zu authentifizieren. In der Regel authentifizieren sich nur die Server. Der Inhalt der Datenübertragung wird dann mit dem ausgehandelten Schlüssel symmetrisch verschlüsselt. Um eine höhere Sicherheit gewährleisten zu können, werden für jede Übertragung neue Sicherheitsparameter ausgehandelt. Die Nachrichtenintegrität wird durch Message Authentication Codes (MAC)¹¹ gewährleistet. Von Zertifizierungsstellen (Certificate Authorities) signierte Zertifikate dienen als Nachweis der Identität.

2.2.2. Sicherheit

Obwohl *TLS 1.2* als hinreichend sicher gilt, steht die Sicherheit von TLS unter Kritik. Die meisten Browser unterstützen bislang nur die unter anderem durch *BEAST attacks*¹² angreifbare Version *TLS 1.0*. Als Grund für die geringe Verbreitung von *TLS 1.2* nennt [Rit12] den zu großen Implementierungsaufwand für einen zu geringen Mehrwert. Selbst wenn *TLS 1.2* benutzt werden soll, ist es möglich die Sicherheitslücken von *TLS 1.0* mithilfe einer *downgrade attack* auszunutzen, da bei einem gescheiterten Handshake auf *TLS 1.0* zurückgegriffen wird.

Neben dem Nachteil der geringen Verbreitung von *TLS 1.2* stehen vorwiegend die Zertifizierungsstellen unter Kritik. Immer wieder gelingt es Angreifern in Zertifizierungsstellen einzubrechen und sich Zertifikate für beliebige Domains auszustellen.¹³ Der Besitz eines solchen Zertifikats ermöglicht Angreifern unter anderem das Abhören und Verfälschen von TLS gesicherten Verbindungen. Desweiteren wird bemängelt, dass einige Zertifizierungsstellen ihre Ausstellungskriterien zu locker handhaben und die *chain of trust* grundsätzlich kaputt ist.¹⁴

2.3. Active Directory

Verzeichnisdienste lagern, organisieren und stellen Informationen in einem Netzwerk zur Verfügung. **Active Directory (AD)** ist ein von Microsoft Windows Server und Samba benutzter Verzeichnisdienst. Er kann benutzt werden um ein Netzwerk und seine

¹¹vgl. [TW10] Section 8.6.1

¹²vgl. [DR11]

¹³vgl. [AE12, Section 3]

¹⁴vgl. [SS12]

Benutzer zu gliedern und Zugriffsbeschränkungen festzulegen.¹⁵ Das Informatik Rechenzentrum der Universität Hamburg benutzt ein *Active Directory* zur Verwaltung seiner Benutzerkonten.

Um Informationen aus einem Verzeichnisdienst abzurufen, wird ein Protokoll benötigt. Über das **Lightweight Directory Access Protocol (LDAP)** können Informationen aus einem Active Directory abgerufen werden. LDAP wird im OSI-Modell in der Anwendungsschicht angesiedelt. Die aktuellste Version *LDAPv3* erschien im Dezember 1997¹⁶ und wurde im Juni 2006 mit [RFC5246] revidiert.

¹⁵vgl. [Po09, Chapter 1]

¹⁶vgl. [RFC2251]

3. OAuth

In diesem Kapitel wird **OAuth** vorgestellt und in einen Zusammenhang zum Web 2.0 gebracht. Nachdem eine simple Authentifizierungsmöglichkeit skizziert und bewertet wurde, werden *OAuth 1* und *OAuth 2* verglichen, der Autorisierungsablauf von *OAuth 2* erklärt und die Sicherheit von OAuth diskutiert.

Das World Wide Web bildet einen sich schnell verändernden Raum. Mit zunehmender Akzeptanz und Nutzung von Social Media¹⁷ wird eine für den Benutzer einfache, einheitliche und sichere Autorisierungs- und Authentifizierungsmöglichkeit immer wichtiger.

Vor dem großen Web-2.0-Boom um die Jahrtausendwende waren Internet-Nutzer überwiegend Konsumenten, weshalb eine Authentifizierung und Autorisierung für die meisten Anwendungsgebiete unüblich war. Heutzutage ermöglichen viele Anwendungen¹⁸ es dem Benutzer mit dem Web Portal seiner Wahl zu interagieren. Benutzer können Inhalte und Dienste dieser Anwendungen mit ihrem Benutzerkonto (Account) bei dem Web Portal benutzen, aufrufen, kommentieren, teilen und mögen.

Bei OAuth handelt es sich um ein Protokoll, welches eine *Autorisierung dritter Anwendungen* im Auftrag des Benutzers ermöglicht. Die größten Web-Portale stellen ihren Benutzern eine OAuth-Autorisierung zur Verfügung.¹⁹ Dem Benutzer wird eine vertraute Oberfläche an einer ihm bekannten Adresse geboten, auf der er einer dritten Anwendung Zugriffsrechte auf seine Ressourcen einräumen kann. In der Praxis bieten Anwendungen Benutzern zunehmend die Möglichkeit sich durch OAuth mit ihrem Facebook-, Google- oder Twitter-Account zu authentifizieren. Das macht eine Registrierung bei den Anwendungen und somit das Teilen sensibler Geheimnisse mit ihnen obsolet.

3.1. Authentifizierung im World Wide Web

Um sich im World Wide Web zu authentifizieren muss man ein Geheimnis mit der authentifizierenden Stelle teilen und es an diese übermitteln. Ohne Authentifizierungsdienste müsste jede Anwendung ein eigenes Authentifizierungsverfahren implementieren. Bei jeder Implementierung können potentiell Fehler entstehen, durch die eine Anwendung unsicher wird. Häufige Fehlerquellen bilden eine unausreichend verschlüsselte Übertragung *und* Lagerung der Credentials. Da Authentifizierungsvorgänge für Benutzer intransparent sind, müssen sie jeder Anwendung an die sie ihre Credentials übermitteln, vertrauen. Von einem zentralisierten Authentifizierungs- und Autorisierungsdienst profitieren somit sowohl Entwickler als auch Benutzer von Webanwendungen.

¹⁷vgl. [KH10] Section 1

¹⁸neben Webanwendungen auch zunehmend kompilierte Software

¹⁹vgl. Wikipedia: List of OAuth Service Providers. http://en.wikipedia.org/wiki/OAuth#List_of_OAuth_service_providers (Zugriff am 26.02.2013)

3.2. Versionsgeschichte

Das OAuth Protokoll wurde von einer kleinen Gruppe von Webentwicklern geschaffen. Die erste Version *OAuth 1.0* erschien im Oktober 2007. Aufgrund einer Sicherheitslücke²⁰, welche eine *session fixation attack*.²¹ erlaubte, erschien im Juni 2009 eine überarbeitete Version *OAuth 1.0a*. Das im April 2010 erschienene [RFC5849] bildet eine Spezifikation von *OAuth 1*.

Aufgrund der komplizierten und aufwendigen Implementierung von *OAuth 1* wurde das Protokoll von der IETF überarbeitet. Mit dem im Oktober 2012 erschienenen [RFC5849] wurde eine zur ersten Version inkompatible, komplett überarbeitete Spezifikation für *OAuth 2* festgelegt. Statt auf eine Signierung der Token setzt *OAuth 2* auf eine TLS verschlüsselte Verbindung. Dadurch wird vor allem das Erlangen und der Austausch von Zugriffstokens vereinfacht.

3.3. OAuth 2 Workflow

Der OAuth 2 Workflow wird in [RFC5849] spezifiziert. [Leb11, Chapter 9] bietet eine anschauliche, [Boy12] eine ausführlichere Erklärung. Bevor man sich dem *OAuth 2* Autorisierungsablauf widmen kann, gilt es einige Begrifflichkeiten zu klären.

Der Autorisierungsablauf wird als **3-legged OAuth** bezeichnet. Die „**drei Beine**“ stehen für den **Benutzer (User)**, den **OAuth-Server** und eine **Client-Anwendung**. Das OAuth Protokoll wird wirksam, wenn ein Benutzer zugriffsgeschützte²² Dienste einer Client-Anwendung nutzen möchte.

Der Zugriff auf Ressourcen erfolgt über HTTP-Requests nach **Representational State Transfer (REST)**.²³ Alle Ressourcen werden in der **JavaScript Object Notation (JSON)**²⁴ zurückgegeben. Client-Anwendungen müssen sich mit **Bearer Tokens**²⁵ gegenüber dem OAuth-Server authentifizieren. Im wesentlichen kennt OAuth drei verschiedene Token Typen:

auth_code

Authorization Codes werden vom Server ausgestellt und über den Benutzer an eine Client-Anwendung übertragen. Diese tauscht den Authorization Code bei dem Server gegen ein Access Token.

²⁰vgl. OAuth Security Advisory: 2009.1. <http://oauth.net/advisories/2009-1/> (Zugriff am 27.02.2013)

²¹eine Übernahme der Session

²²Die Benutzung setzt eine Authentifizierung und gegebenenfalls Autorisierung für den Zugriff der Benutzer-Ressourcen voraus.

²³vgl. [Fie00, Chapter 5]

²⁴vgl. [Cro06]

²⁵vgl. Abschnitt 3.4

access_token

Access Tokens autorisieren Client-Anwendungen zum Zugriff auf die Ressourcen des Benutzers. Access Tokens sind an ein vom Benutzer bestimmtes **Scope** gebunden. Ein Scope definiert auf welche Ressourcen-Zugriff gewährt wird. Access Tokens werden *direkt* vom Server an Clients ausgeteilt.

refresh_token

Refresh Tokens können von Client-Anwendungen gegen Access Tokens getauscht werden. Da sie langlebig sind, ermöglichen sie Client-Anwendungen einen längeren Zugriff auf Benutzer-Ressourcen. Der Einsatz von Refresh Tokens ist optional. Falls ihr Einsatz vorgesehen ist, werden sie zusammen mit Access Tokens vom Server ausgeteilt.

Aus Sicherheitsgründen müssen sich Client-Anwendungen bei jeder Anfrage mit einem **Client Password (client_secret)** beim Server authentifizieren. Dadurch wird Angreifen bei Man-in-the-middle-Angriffen²⁶ erschwert, an ein Access-Token und Benutzer-Ressourcen zu kommen.

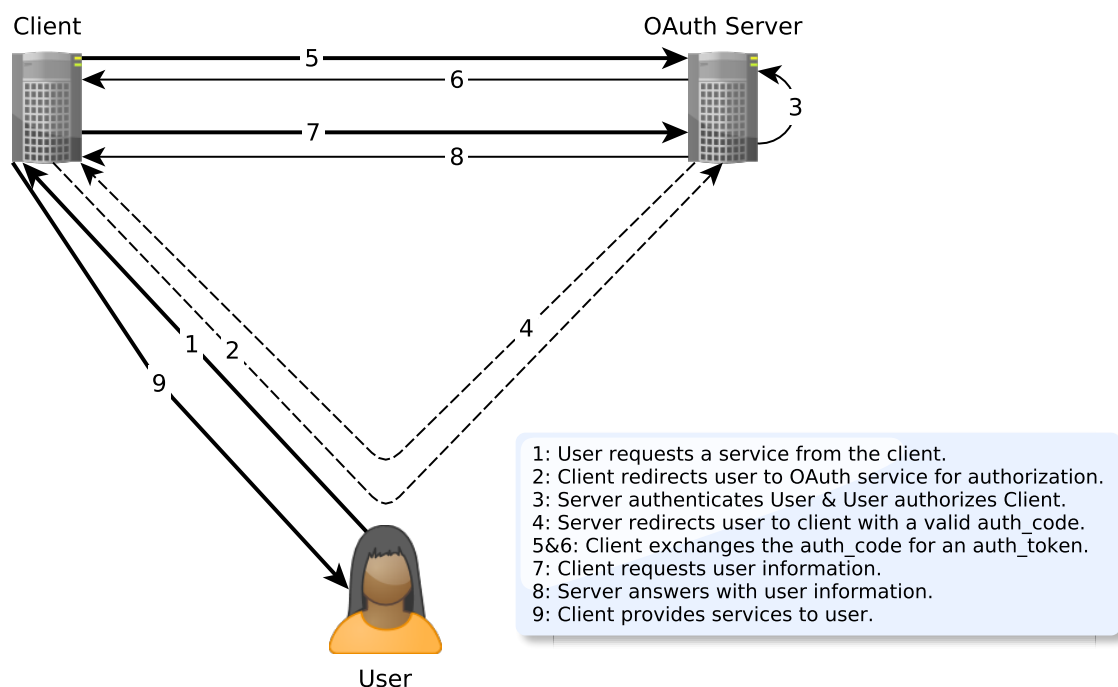


Abbildung 1: 3-legged OAuth

Ein üblicher *OAuth 2* Autorisierungsablauf, wie in Abbildung 1 skizziert, wird durch einen Benutzer eingeleitet, der auf geschützte Dienste einer Client-Anwendung zugrei-

²⁶vgl. Abschnitt 4.5.5

fen möchte ①. Der Benutzer wird auf die zur Client-Anwendung gehörige *Autorisierungsseite* auf dem OAuth-Server weitergeleitet ②. Dort muss er sich, falls noch nicht geschehen, authentifizieren und der Client-Anwendung Zugriffsrechte zugestehen ③. Aus Transparenzgründen soll der OAuth-Server dem Benutzer *vor* einer Autorisierung das beanspruchte Scope offenlegen. Nach einer erfolgreichen Autorisierung wird der Benutzer auf eine, von der Client-Anwendung in ② festgelegten, Seite des Clients weitergeleitet ④. Bei dieser Weiterleitung wird ein *Authorization Code* als Variable im Request übergeben. Diesen tauscht die Client-Anwendung beim OAuth-Server gegen ein *Access Token* ⑤ & ⑥. Nach einem erfolgreichen Token-Tausch hat der Benutzer gegenüber der Client-Anwendung bewiesen, dass er einen gültigen Account bei dem OAuth-Server besitzt. Falls eine *anonyme* Benutzung vorgesehen ist, kann die Client-Anwendung nach einem *erfolgreichen* Token-Tausch mit ⑨ fortfahren. Mit dem *Access Token* kann die Client-Anwendung nun auf die Ressourcen des Benutzers zugreifen ⑦ & ⑧. Letztendlich kann die Client-Anwendung dem Benutzer ihre Dienste zur Verfügung stellen ⑨.

3.4. Sicherheit

Da es sich bei OAuth um eine Protokollspezifikation handelt, hängt die Sicherheit des Systems *maßgeblich* von der jeweiligen Implementierung ab. Mit *OAuth 2* wurde, in einem eigenen [RFC6750], auch die Benutzung von **Bearer Tokens** vorgestellt. Bearer Token sind eindeutige Zeichenketten die anstelle von *Credentials* benutzt werden um auf Ressourcen zuzugreifen. Authorization Codes, Authorization Tokens und Refresh Tokens können als Bearer Tokens realisiert werden. Der Besitz eines Bearer Token autorisiert den Zugriff auf die damit verbundenen Ressourcen. Aus Sicherheitsgründen werden sie durch regelmäßige Invalidierung kurzlebig gehalten. Durch ihre Kurzlebigkeit bringt der Besitz eines Tokens einem Angreifer nur kurzzeitig einen Vorteil. Aus diesem Grund sind sie bei einer Authentifizierung statischem „Wissen“²⁷ vorzuziehen. Bearer Token müssen bei *jedem* Request übertragen werden und sind *nicht* signiert, weshalb sämtliche Kommunikation *zwingend* TLS verschlüsselt erfolgen muss. Da *OAuth 2* vorrangig auf TLS Verschlüsselung setzt, ist es maximal so sicher wie TLS selbst.

Kurz vor Fertigstellung der *OAuth 2* Spezifikation, verkündete der Hauptautor Eran Hammer seinen Rücktritt. Diesen begründet er vor allem mit einer geringeren Sicherheit gegenüber der Vorgängerversion.²⁸ Das Kernproblem bestehe in einem unüberbrückbaren Konflikt zwischen der Web- und Konzern-Welten. Trotz seiner Kritik räumt er ein, dass *OAuth 2* „in den Händen eines Entwicklers mit tiefgründigem Verständnis der Web-Sicherheit“ zu einer sicheren Implementierung führen kann.

²⁷vgl. [WK06] Hannes Federrath, Andreas Pfitzmann: „IT-Sicherheit“. Kapitel 2.2 (S. 276-278)

²⁸vgl. Eran Hammer: OAuth 2.0 and the Road to Hell. <http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/> (Zugriff am 01.03.2013)

4. OAuth InfRZ

4.1. Anforderungen

Die Implementierung eines Single-Sign-On-Dienstes für den Fachbereich Informatik der Universität Hamburg muss *sicher, vertrauenswürdig, benutzerfreundlich, selbsterklärend, wart- und erweiterbar* sein. Neben den eben genannten Eigenschaften sollte eine **Client-Library** existieren.

*„What are the attributes of good software?
Good software should deliver the required functionality and performance to the user
and should be maintainable, dependable, and usable.”*

– [Som10] Section 1.1

Die **Sicherheit von Webanwendungen** ist ein zentraler Faktor in der Webentwicklung. Eine sichere Anwendung lässt keinen unautorisierten Zugriff auf ihre Daten zu.²⁹ Sie umfasst alle zur Laufzeit der Anwendung getroffenen Maßnahmen zur Einhaltung der *Sicherheitsleitlinie*. Eine Web-Anwendung ist *hinreichend* sicher, wenn Maßnahmen gegen alle bekannten Angriffsszenarien getroffen wurden.

Eine **vertrauenswürdige Webanwendung** nutzt die Informationen eines Benutzers *nicht* anders als von ihm erwartet und gewollt. Als Authentifizierungs- und Autorisierungsdienst müssen Benutzer OAuth InfRZ zwingend sensible Credentials anvertrauen. Mit der offenen Verfügbarkeit des Quellcodes werden die Prozesse einer Anwendung transparenter und somit vertrauenswürdiger.

Eine **benutzerfreundliche Webanwendung** funktioniert intuitiv und bietet den Benutzern bekannte Elemente. Benutzer haben in der Regel wenig Motivation sich mit komplizierten Prozessen zu beschäftigen, weshalb es wichtig ist Webanwendungen so benutzerfreundlich wie möglich zu gestalten. Das *Webdesign (Gestaltung, Aufbau und Nutzerführung)* entscheidet maßgeblich über die *Benutzerfreundlichkeit (Usability)*. Eine höhere Usability kann durch den Einsatz von verbreiteten Frameworks³⁰ erzielt werden.

Bei **selbsterklärenden Webanwendungen** werden Benutzer stets über den Zweck der aktuell besuchten Seite informiert und es werden ihnen ihre Optionen aufgezeigt. Selbsterklärende Webanwendungen sind benutzerfreundlicher und schaffen durch die dazugewonnene Transparenz Vertrauen.

Eine **wart- und erweiterbare Webanwendung** lässt sich einfach an neue Anforderungen anpassen. In Anbetracht der Tatsache, dass der Autor das Projekt mit der Fertigstellung an das Rechenzentrum übergibt, ist es umso wichtiger die Anwendung für andere Entwickler verständlich zu implementieren. Eine hohe Wartbarkeit erreicht man unter

²⁹vgl. [Som10] Section 10.1.1

³⁰vgl. [Som10] Chapter 16

anderem durch eine gute Dokumentation und eine sinnvolle Benutzung von verbreiteten Programmierparadigmen und Entwurfsmustern.³¹

Eine gut dokumentierte *Client-Library* bietet Entwicklern eine leicht verständliche Schnittstelle für den Zugriff auf OAuth InfRZ. Sie bietet eine fertige Implementierung aller Prozessabläufe, die durch Methodenaufrufe steuerbar sind. Entwickler müssen sich dadurch nicht mehr mit den Spezifikationen des Authentifizierungsprozesses beschäftigen. Eine *Client-Library* sollte Informationen in wohlgeformten Datenstrukturen zurückliefern wodurch deren Verarbeitung vereinheitlicht und vereinfacht wird. Die Client-Library muss gut dokumentiert und programmiersprachenunabhängig³² implementierbar sein.

4.2. Benutzung

OAuth InfRZ läuft bereits auf einem Rechner des Rechenzentrums³³ und kann mithilfe der **Demo Anwendung**³⁴ ausprobiert werden. Aus Gründen der Barrierefreiheit³⁵ wird OAuth InfRZ in Englisch angeboten. Übersetzungen in andere Sprachen lassen sich mit mittlerem Aufwand nachrüsten. Die Oberfläche von OAuth InfRZ wurde möglichst schlicht gehalten. Im folgenden wird auf die Benutzung und Oberfläche von OAuth InfRZ eingegangen.

4.2.1. Benutzerrollen

OAuth InfRZ wird von zwei Benutzerrollen benutzt, die unterschiedliche Rechte und Aufgaben besitzen.

Benutzer

Mit **Benutzer** werden die Endnutzer bezeichnet. Sie benutzen OAuth InfRZ um sich gegenüber Anwendungen zu authentifizieren und diese mit dem Zugriff auf ihre Information zu autorisieren. Sie haben Zugriff auf Haupt, Login-, und Autorisierungsseiten.

Client-Moderator

Client-Moderatoren verwalten Client-Anwendungen von OAuth InfRZ. Sie können Client-Anwendungen erzeugen, bearbeiten und löschen. Sie haben Zugriff auf die zum Ressourcen-Zugriff benötigten Credentials – *Client-ID* und *Client-Secret*. Es ist ihre Aufgabe den Entwicklern ihrer Client-Anwendungen die dazugehörigen Credentials zur Verfügung zu stellen.

³¹vgl. [Som10] Section 6.3 & Section 7.2

³²damit Entwickler ihre Client-Anwendung in der Sprache ihrer Wahl schreiben können

³³erreichbar aus dem Informatik internen Netz unter <https://svs-sso.informatik.uni-hamburg.de>

³⁴erreichbar aus dem Informatik internen Netz unter <https://svs-sso.informatik.uni-hamburg.de/Demo>

³⁵vor allem in Rücksichtnahme auf unsere internationalen Studierenden

4.2.2. Menüleiste

Authentifizierte Benutzer werden namentlich begrüßt. So wird sichergestellt, dass der Benutzer zu jedem Zeitpunkt weiß mit welchem Account er authentifiziert ist. Client-Moderatoren wird der zusätzliche Menüpunkt „*Manage clients*“ angezeigt, mit dem sie Zugriff auf das Client-Management-System erhalten. Abbildung 2 zeigt eine vollständige Menüleiste.

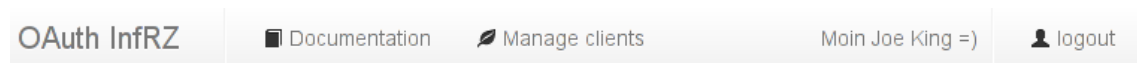


Abbildung 2: Menüleiste

4.2.3. Hauptseite

Die in Abbildung 3 dargestellte Hauptseite bietet eine Erklärung der Funktionalität und fasst alle wichtigen Informationen über den Dienst zusammen. Sie bietet Links zur verwendeten Lizenz, technischen Dokumentation und zum Repository des Quellcodes. Diese Maßnahmen sollen die Vertrauenswürdigkeit fördern.

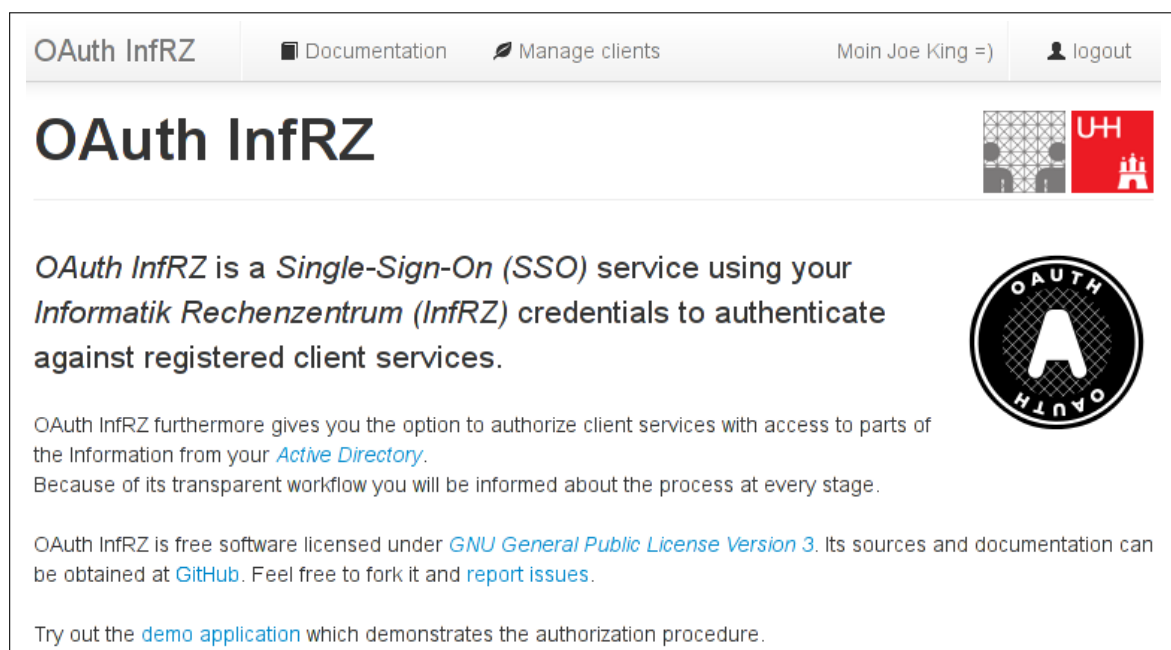


Abbildung 3: Hauptseite

4.2.4. Authentifizierung

Benutzer können sich auf der **Loginseite** gegen das Active Directory authentifizieren. Die Eingabefelder der Loginseite sind selbsterklärend. Üblicherweise gelangen Benutzer über die Autorisierungsanfrage eines Clients auf die Loginseite. In diesen Fällen wird, wie in Abbildung 4 illustriert, eine farblich hervorgehobene Info-Box eingeblendet in welcher der Client genannt wird. Außerdem wird der Benutzer nach einer erfolgreichen Authentifizierung zur zum Client gehörigen Autorisierungsseite weitergeleitet.

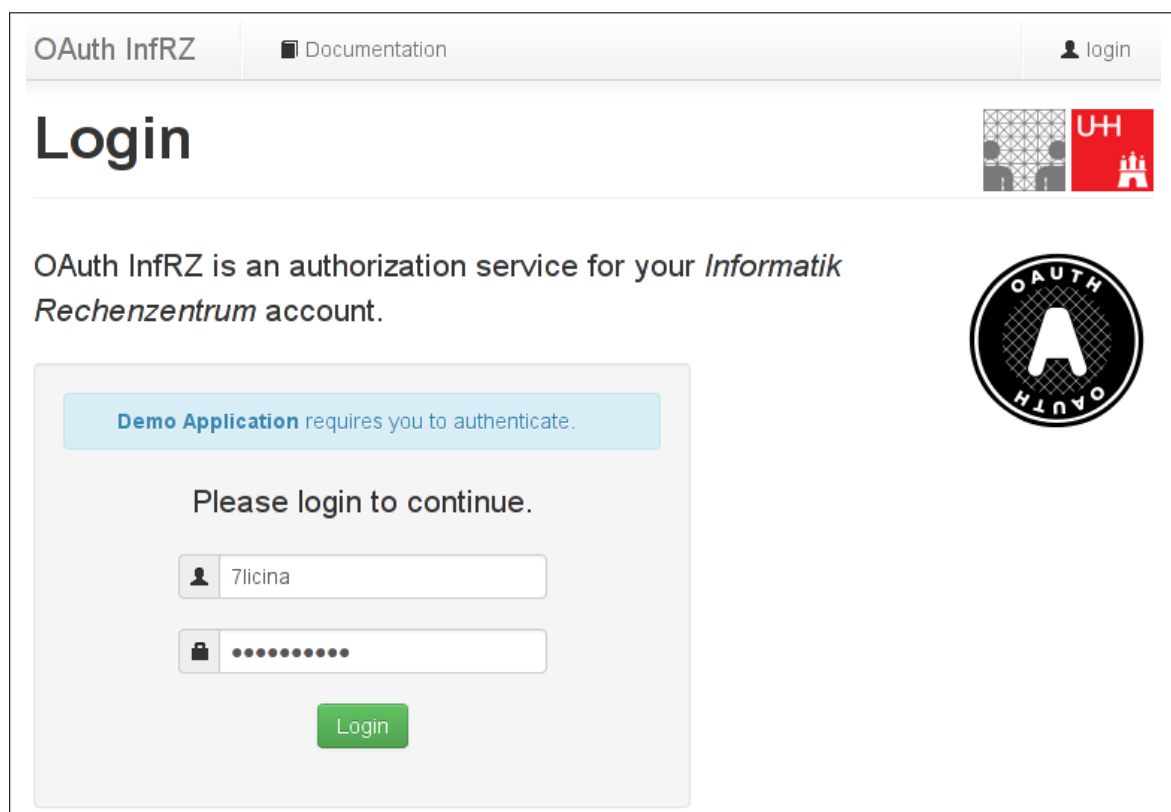


Abbildung 4: Loginseite: Authentifizierungsaufforderung von „Demo Application“

4.2.5. Autorisierung

Um einen Client-Dienst zu nutzen, müssen Benutzer diesen erst autorisieren. Dies geschieht über die in Abbildung 5 gezeigte **Autorisierungsseite** des Clients. Der Benutzer legt dabei fest, auf welche seiner Informationen die Client-Anwendung über OAuth InfRZ zugreifen darf. Diese Seite stellt Informationen zur Client-Anwendung und dem angeforderten Scope bereit. Ein Client-Moderator legt das von einer Client-Anwendung angeforderte Scope fest. Dabei kann der Client-Moderator sowohl erforderliche (required) als auch optionale Zugriffsrechte beantragen. **Optionale Felder** können Benutzer

abwählen und dem Client somit den Zugriff auf die dazugehörige Ressource verwehren. **Erforderliche Felder** hingegen werden ausgegraut angezeigt und können nicht ausgewählt werden. Nach einer erfolgreichen Autorisierung wird der Benutzer zu einer vom Client bestimmten URL des Clients weitergeleitet.

The screenshot shows the OAuth authorization interface for a 'Demo Application'. At the top, there is a navigation bar with 'OAuth InfRZ', 'Documentation', 'Manage clients', and a user profile 'Moin Joe King =)' with a 'logout' link. The main heading is 'Demo Application' next to a grid icon and a red 'UH' logo. The message states: 'Demo Application is requesting you to authenticate.' Below this, it says: 'By clicking on "Agree" you are authorizing Demo Application to access the below selected information.' A light blue box contains the text: 'Demo Application: This application is completely redundant!'. The authorization details section, titled 'You are authorizing access to:', lists four permissions, all with checked checkboxes: 'Kennung' (Required for identification), 'Name' (Needed if you want to use our services with your name), 'E-Mail' (Required for communication), and 'Groups' (Required for authentication). At the bottom, there is a green 'Agree' button and the text: 'You will be redirected back to Demo Application.'

Abbildung 5: Autorisierungsseite für „Demo Application“

4.2.6. Client-Management

Das **Client-Management** ist ausschließlich für Client-Moderatoren einsehbar. Clients können von einem Client-Moderator angelegt und verwaltet werden. Technisch sind Moderatoren Benutzer mit besonderen Rechten. Für die Benutzung des Client-Managements wird eine funktionierende JavaScript-Engine benötigt. Client-Moderatoren sollen im laufenden Betrieb vom Rechenzentrum verwaltet werden.³⁶

³⁶eine ausführliche Diskussion zu diesem Thema gibt es in Abschnitt 4.8.3

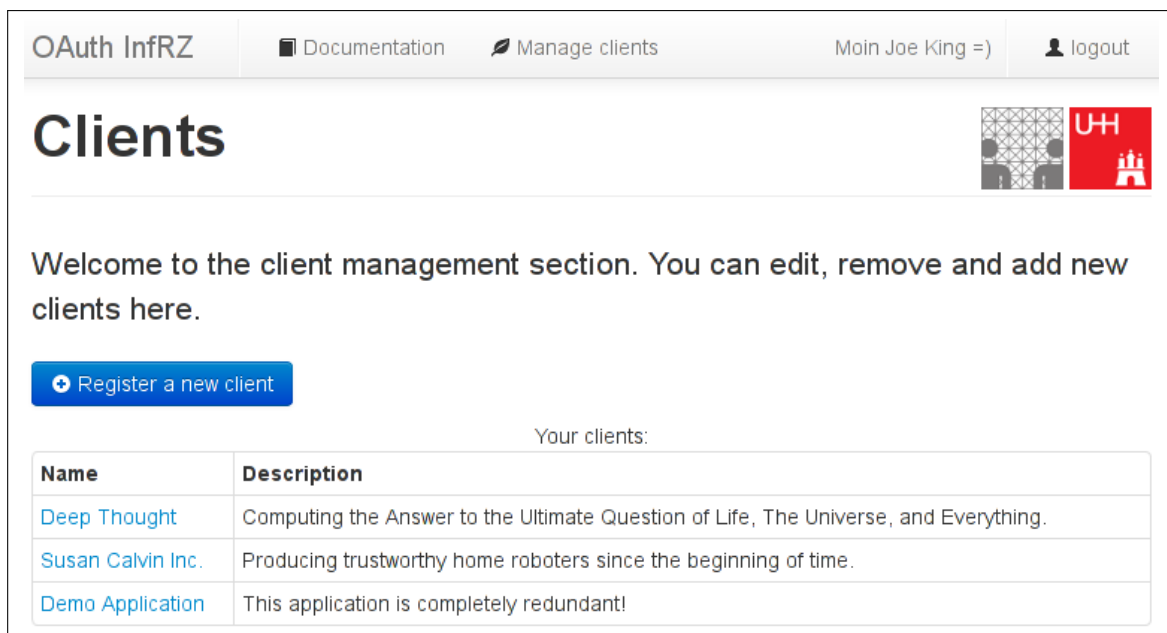


Abbildung 6: Client-Übersichtsseite

Über den Menüpunkt „*Manage clients*“ gelangt ein Moderator auf die in Abbildung 6 illustrierte *Client-Übersichtsseite*. Auf dieser Seite werden alle vom Moderator verwalteten Clients aufgelistet. Von hier aus gelangt man über den „*Register a new client*“-Button zur in Abbildung 7 gezeigten **Client-Registrierungsseite**. Die Namen in der Tabelle sind Links zur entsprechenden **Clientseite** (Abbildung 8).

Auf der **Client-Registrierungsseite** können neue Clients bei OAuth InfRZ registriert werden. Die Client-Registrierungsseite bietet ein HTML-Formular zur Einstellung der Client Eigenschaften. Nicht sofort ersichtliche Felder werden mit **Tooltips**³⁷ erklärt. Der „**Name**“ soll für den Dienst der dazugehörigen Webanwendung *passend* gewählt werden. Am besten entspricht er dem in der Webanwendung selbst benutzten Namen. Aus der **Beschreibung („Description“)** soll ersichtlich sein, was dieser Dienst macht *und* weshalb er eine Autorisierung vom Benutzers erfordert. Als „**Host**“ ist der *Hostname* oder die *IP-Adresse* des Dienstes einzutragen. Eine Zugriffsbeschränkung der Requests über die Adresse des Client Servers wird in Abschnitt 4.8.2 diskutiert. Als „**Redirect URI**“ ist die zur Verarbeitung des Authorization Codes vorgesehene Seite der Client-Anwendung anzugeben. Felder zu nicht als *verfügbar* („*available*“) markierten „**Scopes**“ sind deaktiviert und werden ausgegraut dargestellt. Die „**Scope info**“ wird auf der Autorisierungsseite als Erklärung zum Scope angezeigt. Sie soll Auskunft darüber geben, weshalb der Dienst Zugriff auf diese Information benötigt.

Neben einer Auflistung aller einstellbaren Eigenschaften, beinhalten Clientseiten die Credentials (die „*Client-ID*“ und das „*Client-Secret*“). Bei beiden Werten handelt es sich

³⁷vgl. TechTerms: Tooltip. <http://www.techterms.com/definition/tooltip> (Zugriff am 15.03.2013)

OAuth InfRZ
Documentation
Manage clients
Moin Joe King =)
logout

Register a new client

Name

Description ⓘ

SVS-Blog is the weblog of SVS.
You need to authenticate for an authorization proof.

Host ⓘ

Redirect URI ⓘ

Scope ⓘ

The user must grant access to this information to proceed.

scope	available ⓘ	required ⓘ	info ⓘ
kennung	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Required for authorization.
name	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Optionally needed for comfort.
email	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>
groups	<input type="checkbox"/>	<input type="checkbox"/>	<input type="text"/>

+ Register client

Abbildung 7: Client-Registrierungsseite

um eine zufällige Zeichenkette.

Client-ID

Jeder Client besitzt eine eindeutige **Client-ID** anhand der er bei OAuth InfRZ identifiziert wird.

Client-Secret

Das **Client-Secret** ist ein Geheimnis, durch dessen Angabe sich ein Client gegenüber OAuth InfRZ authentifiziert. Die Betreiber einer Client Anwendung müssen es angemessen vor unbefugtem Zugriff schützen.

OAuth InfRZ
Documentation
Manage clients
Moin Joe King =)
logout

Demo Application

Name
Demo Application
Delete
Edit

Description
This application is completely redundant!

Host
https://localhost/

Redirect URI
https://localhost/Demo

Scope

scope	required	info
kennung	yes	Required for identification
name	no	Needed if you want to use our services with your name
email	yes	Required for communication
groups	yes	Required for authentication

Client-ID
246586fbfa81d5b73503a0e811668b177196775ccdecc308bbe5464b798a1bbc36a1c2048380058466c488e60bc3700640aa5f10deff6a2deb176be062d60b3a

Client-Secret ⓘ
ee4037c1077f5657f22db6a08b0d01f07500a26d7b0894c015305c9ba8d6b4248880fb379b1f3ec934ff5bef567b538d9eae19c29a0246cf7523edf2672596b3

Renew Client-ID & Client-Secret ⓘ

Abbildung 8: Clientseite

Clientseiten bieten die Optionen den Client zu löschen, zu editieren und sich neue Credentials zuweisen zu lassen. Vor dem Löschen eines Clients wird eine Löschbestätigung angezeigt. Die Oberfläche zum Bearbeiten des Clients ähnelt der Client-Registrierungsseite.

4.3. Eingesetzte Werkzeuge

Als **Werkzeuge** werden in diesem Abschnitt alle zur technischen Umsetzung benutzten Programme, Programmiersprachen und Frameworks bezeichnet. Alle eingesetzten Werkzeuge müssen den Entwickler dabei unterstützen die an die Anwendung gestellten Anforderungen umzusetzen. Grundsätzlich gilt es für das Anwendungsgebiet angemessene Werkzeuge einzusetzen. Werkzeuge sind angemessen, wenn sie eine saubere, hinreichend performante Lösung für das vorgesehene Anwendungsgebiet ermöglichen. Um eine unnötige Komplexität zu verhindern, gilt es vor dem Einsatz einer neuen Technologie gründlich abzuwägen, ob ihr Nutzen den damit zusammenhängenden Mehraufwand rechtfertigt.

4.3.1. Back-End: PHP 5.4, SQLite, nginx

Zum **Back-End** gehören in der Webentwicklung alle Prozesse, die *serverseitig* ausgeführt werden.

PHP: Hypertext Preprocessor (PHP) ist eine *serverseitig ausgeführte Skriptsprache*. Es ist die verbreitetste³⁸ Programmiersprache für Webanwendungen. PHP ist imperativ, objektorientiert und frei. PHP erschien erstmals 1995, seither wurde es stetig korrigiert, erweitert und verbessert. Die *PHP-Syntax* ist an verbreitete Programmiersprachen wie C, C++, Java oder Perl angelehnt, weshalb PHP für erfahrene Entwickler leicht erlernbar ist. Zusätzlich zu zahlreichen *PHP-Libraries*³⁹ liefert das WWW eine Vielzahl von Erweiterungen, Anleitungen und Beispielen, die Entwickler unterstützen. Seit *PHP 5.3* liegt der Fokus primär auf der *Objektorientierung*⁴⁰, welche eine moderne und saubere Programmierung ermöglicht. PHP-Entwickler schätzen die schwache, dynamische Typisierung, die eine agile Entwicklung unterstützt. Auf <http://php.net> befindet sich eine ausführliche Dokumentation mit Beispielen und Benutzer-Kommentaren.

Aufgrund einer großen Community und der damit einhergehenden Erfahrung sollte PHP bei der Entwicklung von Webanwendungen in Erwägung gezogen werden. PHP liefert Libraries für alle Kernfunktionalitäten von OAuth InfrZ, weshalb es sich zur Benutzung eignet.

SQLite ist ein *relationales Datenbanksystem*. Als eingebettetes Datenbanksystem⁴¹ ist der Anwendungsbereich von SQLite nicht auf Server begrenzt, vielmehr findet man es in

³⁸vgl. w3techs: Usage of server-side programming languages for websites. http://w3techs.com/technologies/overview/programming_language/all (Zugriff am 04.03.2013)

³⁹vgl. Wikipedia: List of PHP libraries. http://en.wikipedia.org/wiki/List_of_PHP_libraries (Zugriff am 04.03.2013)

⁴⁰vgl. Heise Developer: PHP 5.3 mit vielen neuen Funktionen. <http://www.heise.de/developer/meldung/PHP-5-3-mit-vielen-neuen-Funktionen-187509.html> (Zugriff am 04.03.2013)

⁴¹vgl. SQLite: Most Widely Deployed SQL Database. <http://www.sqlite.org/mostdeployed.html> (Zugriff am 04.03.2013)

den Programmen von zahlreichen Endgeräten. SQLite ist public domain und unterstützt einen Großteil des in [SQL1992] spezifizierten *SQL92-Standard*. Anstatt seine Informationen auf mehrere Dateien zu verteilen, speichert SQLite die komplette Datenbank in einer Datei, was die Komplexität verringert und eine Datensicherung vereinfacht. SQLite zeichnet eine einfache Handhabung und geringe Größe⁴² aus. Die Wahl des Datenbanksystems ist maßgeblich von dem Einsatzszenario abhängig. Bei einer Webanwendung mit verschwindend geringer Last *und* überschaubaren Datenmengen, wie OAuth InFRZ, ist SQLite eine gute Wahl.

nginx ist ein freier *high-performance Webserver*. Er wird für seine hohe Performance, gute Skalierbarkeit, simple Konfiguration und Ressourcen-Sparsamkeit geschätzt und findet zunehmend Anwendung.⁴³ Obwohl nginx den Anspruch hat sich auf die wesentlichen Funktionen eines Webserver zu beschränken, bietet es Administratoren alle wichtigen Konfigurations-, Debugging- und Maintenanceeigenschaften.

4.3.2. Front-End: HTML5, CSS3, JavaScript

Mit **Front-End** werden alle beim Nutzer angezeigten Ausgaben bezeichnet. Es bildet das Gegenstück zum *Back-End*.

Die **Hypertext Markup Language (HTML)** ist die meistverwendete *Auszeichnungssprache* im WWW. Mit HTML werden die Elemente einer Webseite definiert und in einen Gesamtkontext gebracht. Seit dem im November 1995 erschienenen [RFC1866] hat sich HTML stetig weiterentwickelt und prägt Look and Feel des WWW maßgeblich.

Cascading Style Sheets (CSS) ist eine *Auszeichnungssprache*. Sie werden überwiegend als Stilvorlagen für HTML und andere XML-Dokumente verwendet. Im Gegensatz zu HTML wird CSS ausschließlich für die Gestaltung verwendet. CSS erlaubt dem Entwickler das Erscheinungsbild vom Inhalt zu trennen. Es existieren zahlreiche CSS-Frameworks die einem Entwickler bei der Implementierung des Designs unterstützen.

JavaScript (JS) ist eine weit verbreitete, dynamisch typisierte und objektorientierte *Skriptsprache*. Ursprünglich wurde JS entwickelt um clientseitig im Webbrowser ausgeführt zu werden. Es wird hauptsächlich verwendet um Webanwendungen dynamisch zu gestalten und das **Document Object Model (DOM)** zu manipulieren. Heutzutage findet JS in weiteren Anwendungsgebieten⁴⁴ Verwendung. Bis zur Erscheinung von V8, einer freien von Google entwickelten *JS-Engine*, im September 2008 wurde JS zur Laufzeit interpretiert. Durch den Einsatz von *just-in-time* Kompilierung⁴⁵ erzielte V8 deutlich überlegene Ausführungszeiten⁴⁶ als alle bis dato verwendeten JS-Engines. Mittlerweile haben auch andere Browser auf *just-in-time* Kompilierung umgestellt.

⁴²je nach Betriebssystem zwischen etwa 250 und 300 KiB

⁴³vgl. [Ree08] Abstract

⁴⁴zum Beispiel als Programmiersprache für Webanwendungen mit dem Framework Node.js

⁴⁵just-in-time-Kompilierer übersetzen Programme zur Laufzeit in Maschinencode

⁴⁶vgl. Heise Online: Google Chrome überholt die Konkurrenz. <http://www.heise.de/newsticker/meldung/Google-Chrome-ueberholt-die-Konkurrenz-202963.html> (Zugriff am 04.03.2013)

Die Kombination aus HTML5, CSS3 und JavaScript hat mit der Nachfrage an plattformunabhängigen Programmiermöglichkeiten⁴⁷ an Bedeutung gewonnen. Eine moderne Webanwendung ohne diese Sprachen zu programmieren ist undenkbar.

4.3.3. Frameworks: Twig, Bootstrap, jQuery

Frameworks sind leicht erweiterbare *Programmiergerüste*. Sie unterstützen Softwareentwicklung durch ein Angebot an erprobter Funktionalität. Der Einsatz von verbreiteten Frameworks trägt zu einer besseren Wartbarkeit bei.⁴⁸

Twig ist eine Open Source *template engine für PHP*. Mit Twig lassen sich statische HTML-Seiten aus *dynamischen* Twig-Dateien kompilieren. In *Twig-Dateien* kann man – zusätzlich zu HTML – Kontrollstrukturen, Variablen und Funktionen benutzen um den Inhalt dynamisch anzupassen. Die Einführung von Vererbungsfunktionalität reduziert Code-duplikation und strukturiert den Code durch Kapselung. Der Einsatz von Twig macht das Frontend wartbarer und leichter verständlich.

Bootstrap ist ein freies, von Twitter geschaffenes *CSS Framework*. Bootstrap bietet eine große Auswahl an *Web-Komponenten* für die Benutzeroberfläche. Diese sind aufgrund steigender Beliebtheit vielen Benutzern schon bekannt was eine bessere Usability zu Folge hat.

jQuery ist das meistverwendete⁴⁹ *JavaScript Framework*. jQuery bietet eine große Auswahl an Funktionen mit deren Hilfe man auf DOM-Objekte zugreifen und manipulieren kann. jQuery ist ein Free Software Projekt.

Zu diesen drei Frameworks gibt es ausführliche Dokumentationen mit Code-Beispielen und eine hilfreiche Community.

4.3.4. Verschlüsselung: TLS

Bei der Übertragung von Credentials ist es erforderlich eine angemessene Verschlüsselung zu wählen. Trotz der in Abschnitt 2.2.2 angesprochenen Sicherheitsmängel ist TLS aufgrund seiner weiten Verbreitung zu wählen. Es ist zu erwarten, dass Browser die sicherere Version *TLS 1.2* in naher Zukunft unterstützen werden.

OpenSSL ist eine freie Sammlung von *SSL und TLS tools* für den Server. Mit OpenSSL können Verbindungen mit TLS verschlüsselt werden.

⁴⁷besonders im Smart Phone Sektor

⁴⁸vgl. [Som10] Chapter 16

⁴⁹w3techs: Usage of JavaScript libraries for websites. http://w3techs.com/technologies/overview/javascript_library/all (Zugriff am 05.03.2013)

4.4. Eingesetzte Konzepte

Mit eingesetzten **Konzepten** werden in diesem Abschnitt Programmierparadigmen, Entwurfsmuster und Architekturmuster bezeichnet. Sie dienen der Strukturierung von Code und machen diesen somit wartbarer und leichter verständlich. Für sie gelten analoge Anforderungen wie für Werkzeuge⁵⁰.

Objektorientierte Programmierung (OOP) ist ein weit verbreitetes *Programmierparadigma*. Es beruht auf der Philosophie der Objektorientierung, nach der Teile des Codes ihrem Zweck nach in Klassen geteilt werden. Klassen bilden ein abstraktes Konstrukt und somit Blaupausen für Objekte. Sie beschreiben Funktionen ihrer zugehörigen Objekte. Objekte sind Instanzen von Klassen, die mit anderen Objekten kommunizieren und interagieren. „*The class holds the shared behavior for its instances*“⁵¹. Objektorientierte Programmierung schafft⁵² klare Strukturen und macht den Quellcode dadurch wartbarer und leichter verständlich.

Model View Controller (MVC) ist ein *Entwurfsmuster* zur Gliederung von Code nach seiner Aufgabe. Grundkonzept von MVC ist es durch die Aufteilung der Softwareteile nach dem **Datenmodell (Model)**, der **Benutzerausgabe (View)** und der **Programmsteuerung (Controller)** eine bessere Modularität zu erzielen.⁵³ Durch den Einsatz von MVC gewinnt Code an Struktur und es wird leichter Teile des Codes auszutauschen. Durch den Einsatz von MVC wird eine Anwendung wartbarer und leichter verständlich.

Front-Controller ist ein *Architekturmuster* mit dessen Hilfe sich Adressen auf Controller abbilden lassen. Front-Controller bieten einen zentralen Einstiegspunkt in eine Webanwendung. Alle eingehenden Requests werden zunächst von dem *Front-Controller* bearbeitet. Dieser wählt den zum Request passenden Controller und startet diesen. Front-Controller strukturieren Code und machen ihn übersichtlicher. Dadurch wird Code wartbarer und leichter verständlich.

Autoloader laden benötigte Klassen automatisch nach. Anstatt die vollständige Anwendung zur Initialisierung zu laden, wird bei dem Einsatz eines Autoloaders nur ein minimaler Kern geladen. Durch den Einsatz von Autoloadern werden Anwendungen performanter, da unbenutzte Teile der Anwendung nicht geladen werden.

⁵⁰in der Einleitung zu Abschnitt 4.3 beschrieben

⁵¹vgl. [Kay93] IV.

⁵²im Vergleich zur klassischen prozeduralen Programmierung

⁵³vgl. [Som10] Section 6.3

4.5. Technische Realisierung

4.5.1. Versionierung

Git ist ein freies *verteiltes Versionierungssystem*. Ein Versionierungssystem dokumentiert die Veränderungen von Dokumenten. Ein verteiltes Versionierungssystem benötigt keinen zentralen Server, da es dezentral laufen kann. Das *Git-Repository* von OAuth InfRZ wird auf *GitHub*⁵⁴ weltweit öffentlich zur Verfügung gestellt.

4.5.2. Verzeichnisstruktur und Dateien

Die **Verzeichnisstruktur** von OAuth InfRZ wird maßgeblich vom *MVC-Pattern* bestimmt. Im Folgenden wird auf die einzelnen Verzeichnisse und die darin enthaltenen Dateien eingegangen um einen Kontext dieser Dateien zur Anwendung aufzubauen.

Im Hauptverzeichnis befinden sich die Lizenzinformation (*COPYING*), eine technische Dokumentation (*README.md*), die Konfigurationsdatei (*config.ini*) und der zentrale Einstiegspunkt (*index.php*) in die Anwendung. In der *index.php* werden einige globale Einstellungen getroffen, die Session initialisiert, die Konfiguration geladen und der Front-Controller gestartet.

Das Verzeichnis **Control** enthält neben den eigentlichen Controllern in dem Unterverzeichnis **Modules** auch den Autoloader (*Autoloader.php*) und den Front-Controller (*FrontController.php*). Im Unterverzeichnis **Security** befindet sich das Auth-Factory-Interface (*AuthFactoryInterface.php*) und die LDAP-Auth-Factory (*LDAPAuthFactory*). Implementierungen des Auth-Factory-Interfaces können genutzt werden um den Authentifizierungs-Mechanismus auszutauschen. Dadurch ist OAuth InfRZ mit wenig Aufwand an eine veränderte Infrastruktur anpassbar. Die LDAP-Auth-Factory ist eine Implementierung des Auth-Factory-Interfaces.

Das Verzeichnis **Model** enthält das Datenmodell (jede Klasse in einer eigenen PHP-Datei) und den Database-Wrapper (*DatabaseWrapper.php*). Der Database-Wrapper stellt Funktionen für den Zugriff und die Manipulation der Datenbank zur Verfügung. Jeglicher Zugriff und Manipulation der Datenbank soll über den Database-Wrapper geschehen.

Das Verzeichnis **View** enthält die *twig Templates* (jedes Template in einer eigenen twig-Datei) und den Response-Builder (*ResponseBuilder.php*). Der Response-Builder ist für alle Ausgaben zuständig. Er stellt Funktionen zur Generierung und Rückgabe von Antworten bereit.

Das Verzeichnis **Client** enthält die *Client-Library*. Genauere Informationen zur Client-Library gibt es in Abschnitt 4.7.

⁵⁴vgl. OAuth InfRZ. <https://github.com/Senci/oauth-infrz> (Zugriff am 14.03.2013)

Das Verzeichnis **Demo** enthält eine simple *Client-Anwendung* die zur Demonstration der Funktionalität und Veranschaulichung des Autorisierungsablaufs von OAuth InfRZ dient.

Das **Resources** Verzeichnis enthält alle vom Browser für die Anzeige benötigten Ressourcen. Dazu gehören CSS-Stylesheets (im Unterverzeichnis **css**), JS-Skripte (im Unterverzeichnis **js**) und Bilder (im Unterverzeichnis **img**). Im Unterverzeichnis **bootstrap** befindet sich das Bootstrap Framework.

4.5.3. Front-Controller

Wie schon in Abschnitt 4.4 beschrieben dient ein **Front-Controller** dem Abbilden von Adressen auf Controller. Durch den Einsatz eines Front-Controllers sind Entwickler nicht mehr dazu gezwungen ihre Verzeichnisstruktur nach den URLs zu modellieren. Alle verfügbaren **Aktionen (Actions)** werden im Controller als Methoden deklariert. Bei OAuth InfRZ erkennt nginx die eingehenden Requests nach ihrer URL und bildet diese – sofern sie auf einen der eingetragenen regulären Ausdrücke passt – auf den Front-Controller ab. Informationen über den zuständigen Controller und die aufzurufende Action werden dabei intern in dem Feld „*action*“ mit der Syntax „*Controller_Action*“ übertragen. Intern funktioniert das über einen Redirect, der Benutzer bekommt davon allerdings nichts mit. Der Front-Controller dekodiert das „*action*“-Feld, erstellt dynamisch eine Instanz des entsprechenden Controllers und ruft die entsprechende Action an dem Controller auf. Bei PHP bietet es sich an zu diesem Zweck **variable Variablen**⁵⁵ zu benutzen. Durch variable Variablen lassen sich die tatsächlich aufgerufenen Klassen und Methoden dynamisch zur Laufzeit bestimmen. In Listing 1 (Zeile 17 und 20) wird die Benutzung von variablen Variablen exemplarisch gezeigt. Listing 1 beinhaltet eine aus redaktionellen Gründen vereinfachte Version der **execAction** Methode des Front-Controllers.

⁵⁵vgl. PHP: Variable variables. <http://www.php.net/manual/en/language.variables.variable.php> (Zugriff am 14.03.2013)

```

1 private function execAction($actionCommand)
2 {
3     // extract module name from action command
4     $moduleName = $this->getModuleName($actionCommand) . 'Controller';
5     // extract action name from action command
6     $actionName = $this->getActionName($actionCommand) . 'Action';
7
8     // generate module path & load module
9     $modulePath = getcwd() . '/Control/Modules/%s.php' . $moduleName;
10    require_once($modulePath);
11
12    // generate class name with Namespace
13    $className = 'Infrz\OAuth\Control\Modules\\' . $moduleName;
14
15    // create instance of class with the class name $className
16    /* @var AbstractController $controller */
17    $controller = new $className($this->config, $this->authFactory);
18
19    // run action $actionName
20    $controller->$actionName();
21 }

```

Listing 1: Benutzung von variablen Variablen am Beispiel des Front-Controllers

4.5.4. Auth-Factory

Eine **Auth-Factory** ist eine Hilfsklasse für die Authentifizierung. Die LDAP-Auth-Factory bietet eine Implementation des Auth-Factory-Interface für die LDAP-Authentifizierung am Informatik Rechenzentrum. Sie nutzt die vom Benutzer zur Verfügung gestellten Credentials um auf einen Teil der im Active Directory gespeicherten Informationen zuzugreifen und bildet somit die Kernfunktionalität von OAuth InfrZ. Diese Daten werden nach ihrem Zugriff aufbereitet und als „User“-Objekt zurückgegeben. Die LDAP-Auth-Factory benutzt die von PHP zur Verfügung gestellte *LDAP-Library* um mit dem Active Directory zu kommunizieren. Listing 2 zeigt eine aus redaktionellen Gründen vereinfachte Version der `signIn` Methode.

```

1 public function signIn($username, $password) {
2     $host = 'ldaps://fbidc2.informatik.uni-hamburg.de';
3     $port = 636;
4     // establish link to $host:$port
5     $link = ldap_connect($host, $port);
6     $mail = $username . '@informatik.uni-hamburg.de';
7     // bind to link with user credentials, return false on failure
8     if (!ldap_bind($link, $mail, $password)) {
9         return false;
10    }
11    // set domain name, set filter for search and select fields to access
12    $base_dn = 'dc=informatik,dc=uni-hamburg,dc=de';
13    $filter = 'uid=' . $username;
14    $fields = array('uid', 'sn', 'givenname', 'memberof');
15    // fire the actual search, return false on failure
16    $ldap_result = ldap_search($link, $base_dn, $filter, $fields);
17    if (!$ldap_result or (ldap_count_entries($link, $ldap_result) != 1)) {
18        return false;
19    }
20    // retrieve user information and generate a User Object from it
21    $ldap_user = ldap_get_entries($link, $ldap_result);
22    $ldap_user = $ldap_user[0];
23    ldap_close($link);
24    $user = $this->generateUser($ldap_user);
25
26    return $user;
27 }

```

Listing 2: LDAP-Authentifizierung mit der LDAP-Library von PHP

4.5.5. Sicherheit

TLS verschlüsselte Verbindungen erschweren **Man-in-the-middle**-Angriffe⁵⁶.

Bei Man-in-the-middle-Angriffen übernimmt ein Angreifer die Kommunikation zwischen Systemen, indem er sie abfängt und weiterleitet. Dabei kann er die übermittelten Pakete lesen und modifizieren. Eine solcher Angriff geschieht bei TLS auf den Schlüsselaustausch, um die Pakete entschlüsseln zu können. Ein Angreifer gibt sich dabei jeweils als einer der Kommunikationspartner aus.⁵⁷ Die verschlüsselt übertragenen Pakete kann ein Angreifer nicht verwerten. Zusätzliche Schutzmechanismen erschweren zudem eine Manipulation der Kommunikation. Alle erforderlichen Einstellungen werden in der Konfigurationsdatei der Server-Software getroffen. Ein gültiges Zertifikat kann im Rechenzentrum beantragt werden.

⁵⁶vgl. [RFC6749, Section 10.8]

⁵⁷vgl. [TW10, Section 8.7.2]

PHP Data Objects (PDO) verringern die Anfälligkeit für **SQL-Injections**. Als *Abstraktionsschicht* für den Datenbankzugriff erlaubt PDO eine von der Wahl der Datenbank unabhängige Programmierung. Eine **SQL-Injection** ist die Ausführung von unerwünschten *SQL-Statements*, die ein Eingreifer in seine Eingaben einschleust. Es werden Zeichen mit besonderer Bedeutung im SQL-Kontext verwendet um eine Anweisung einzuschleusen. Die Verwendung von **Prepared Statements** trennt SQL-Befehle von den dazugehörigen Werten, welche beim Binden automatisch „escaped“⁵⁸ werden. Durch konsequenten Einsatz von Prepared Statements kann die Anfälligkeit für SQL-Injections minimiert werden. Listing 3 verdeutlicht exemplarisch die Benutzung von Prepared Statements.

```
1 // initialize PDO
2 $db = new \PDO('sqlite:database_file.sqlite3');
3 $query = 'SELECT item FROM items WHERE color = :color';
4 $statement = $db->prepare($query);
5 // PDO has to fetch the objects as an Object with Class "Model\Car"
6 $statement->setFetchMode(\PDO::FETCH_CLASS, 'Model\Car');
7 // bind parameter ':color' to $oldColor
8 $statement->bindParam(':color', $oldColor);
9 $statement->execute();
10
11 foreach ($statement as $scar) {
12     $scar->changeColor($newColor);
13 }
```

Listing 3: Benutzung von Prepared Statements mit PHP Data Objects

An **Page-Token** gebundene Aktionen verhindern **Cross-site request forgery (CSRF)**. Bei CSRF-Angriffen bringt ein Angreifer den Browser seines Opfers dazu Requests auszuführen. Der Angreifer nutzt dabei offene Sessions des Opfers um Aktionen bei anderen Webanwendungen auszuführen. Ein CSRF-Angriff kann ohne die Kenntnisnahme des Opfers durchgeführt werden. Ein Page-Token ist ein an *einen* Benutzer gebundenes Geheimnis. Es wird üblicherweise als *Hidden Field* in das *HTML-Formular* eingefügt. Für einen Angreifer ist es schwierig Page-Token auszulesen. Da für jede Aktion ein gültiges Page-Token notwendig ist, verhindert deren Benutzung CSRF-Angriffe. Bei OAuth InfrZ sind die Page-Tokens als Bearer Tokens realisiert und werden als *Form-Encoded Body Parameter*⁵⁹ übertragen. Zur Erzeugung wird eine zufällige und eindeutige Zeichenfolge geschaffen, an den Benutzer gebunden und in die Datenbank eingefügt. Listing 4 beinhaltet eine vereinfachte Version des zur Erzeugung eines Page-Token benutzten Codes.

⁵⁸Zeichen mit besonderer Bedeutung werden in eine Escape-Sequenz umgewandelt

⁵⁹vgl. [RFC6750, Section 2.2]

```

1 $p_token = $this->getUniqueHash('page_token', 'token');
2 $insert_token = 'INSERT INTO page_token (user_id, token, expires_at)
3                 VALUES (:user_id, :token, :expires_at);';
4 $stmt = $this->db->prepare($insert_token);
5 $stmt->bindParam('user_id', $user->id);
6 $stmt->bindParam('token', $p_token);
7 $stmt->bindValue('expires_at', time()+(30*60));
8 $stmt->execute();

```

Listing 4: Erzeugung eines Page-Token

Bei der Verifizierung eines Page-Token wird die Benutzerzugehörigkeit und die zeitliche Gültigkeit des Tokens überprüft. Ein Page-Token muss zum aktuell angemeldeten Benutzer gehören und darf nicht älter sein als 30 Minuten. Listing 4 zeigt eine vereinfachte Form des Codes mit dem ein Page-Token Verifiziert wird. Die Funktion `getUniqueHash($tableName, $columnName)`; nutzt den Code aus Listing 6 zur Token-Generierung und sorgt dafür, dass Token für eine Spalte einer Tabelle in der Datenbank eindeutig sind.

```

1 $page_token = $_POST['page_token'];
2 $page_token = $this->db->getPageTokenByToken($page_token);
3 $user = $this->authFactory->getUser();
4 if (!$page_token or !$user or $page_token->user_id != $user->id) {
5     $this->responseBuilder->buildError('no_permission');
6 }
7 if ($page_token->expires_at <= time()) {
8     $e = 'Unfortunately your Page-Token has expired! Please go back, '.
9         'reload the page and try again';
10    $this->responseBuilder->buildError('no_permission', $e);
11 }
12 $this->db->deletePageToken($page_token->token);

```

Listing 5: Verifizierung eines Page-Token

Für Angreifer darf ein Token nicht zu erraten sein, weshalb Token mit einem *guten* Zufalls-Algorithmus erzeugt werden *müssen*. Zur Erzeugung der Token benutzt OAuth InFRZ die seit *PHP 5.3* verfügbare Funktion `openssl_random_pseudo_bytes($length)`; . In [AK12, Section 2.2] wird diese Funktion als „the only function available in order to obtain cryptographically secure random bytes“⁶⁰ beschrieben. Listing 6 verdeutlicht die Erzeugung eines 128 Zeichen langen Tokens.

```

1 $result = base64_encode(openssl_random_pseudo_bytes(96));
2 $result = strtr($result, '+/=', '-_.');

```

Listing 6: Erzeugung eines Tokens

⁶⁰diese Aussage ist nur im PHP-Kontext gemeint

4.6. Verfügbare Requests

Alle zur autorisierung erforderlichen Schritte im *OAuth 2* Protokoll werden in Abschnitt 3.3 erklärt. Eine erfolgreiche autorisierung setzt mehrere HTTP-Requests voraus. Im folgenden werden alle Requests für den in Abbildung 1 skizzierten Workflow für OAuth InfRZ erklärt. Requests an die Client-Anwendung werden exemplarisch an der Demo-Anwendung beschrieben. Client-Anwendungen steht es frei ihre Endpunkte frei zu gestalten, sie können sich jedoch an der Demo-Anwendung orientieren.

Autorisierung eines Clients

Zur autorisierung von Client-Anwendungen müssen die Clientanwendungen den Benutzer zunächst auf die Autorisierungsseite der Client-Anwendung weiterleiten. Dabei müssen die Parameter *client_id* und *redirect_uri* richtig gesetzt sein.

GET <https://my-oauth-server.edu/authorize>

- **client_id**: Die Client-ID der Client-Anwendung.
- **redirect_uri**: Die Redirect-URL, wie in der Konfiguration angegeben.

Die eigentliche Authentifizierung erfolgt durch einen POST-Request. Neben eines gültigen Page-Token müssen *client_id*, *redirect_uri* und *scope* angegeben werden.

POST <https://my-oauth-server.edu/authorize/grant>

- **client_id**: Die Client-ID der Client-Anwendung.
- **redirect_uri**: Die Redirect-URL, wie in der Konfiguration angegeben.
- **scope**: Das vom Benutzer freigegebene Scope.

Authentifizierung eines Benutzers

Damit ein Benutzer autorisieren kann, muss ein Benutzer sich vorerst authentifizieren. Die Loginseite bietet das Interface dazu.

GET <https://my-oauth-server.edu/login>

Eine Authentifizierungsanfrage erfolgt an eine dafür vorgesehene URL. Die Parameter *username* und *password* müssen gültige Werte enthalten, der Parameter *redirect* ist optional.

POST <https://my-oauth-server.edu/login/authorize>

- **username**: Die Kennung des Benutzers.
- **password**: Das zum Benutzer-Account gehörige Passwort.
- **redirect**: Eine URL zu der nach erfolgreichem Login weitergeleitet wird.

Zur Authentifizierung bei einem Client muss der Benutzer dem Client den Authorization Codes übergeben. Dies geschieht durch den Aufruf an die Redirect-URL des Clients. Der Authorization Code wird als Parameter *code* übergeben.

GET <https://client.org/redirect/page>

- **code**: Ein gültiger Authorization Code.

Token-Austausch

Client-Anwendungen können Authorization Codes und Refresh Token bei OAuth InfRZ gegen Access Tokens tauschen. Bei dem Aufruf der entsprechenden URL, müssen die Parameter *grant_type*, *client_id*, *client_secret*, *redirect_uri* und *code* mit gültigen Werten belegt sein. Die Antwort ist ein JSON-Codiertes Objekt. Listing 9 bietet eine Beispieleantwort für ein Access Token.

POST <https://my-oauth-server.edu/authorize/token>

- **grant_type**: Art des benutzten Token: *authorization_code* oder *refresh_token*.
- **client_id**: Die Client-ID der Client-Anwendung.
- **client_secret**: Das Client-Secret der Client-Anwendung.
- **redirect_uri**: Die Redirect-URI, wie in der Konfiguration angegeben.
- **code**: Der Authorization Code oder das Refresh Token.

Ressourcenzugriff

Der Zugriff auf die Benutzer-Ressourcen erfolgt durch einen Request bei dem ausschließlich ein gültiges *access_token* übergeben werden muss. Die Antwort ist ein JSON-Codiertes Objekt. Listing 10 bietet eine Beispielantwort für die Benutzerinformation.

GET <https://my-oauth-server.edu/user>

- **access_token**: Ein gültiges Access Token.

4.7. Client-Library

Die **Client-Library** soll⁶¹ Entwickler von Client-Anwendungen durch das Bereitstellen einer einfach benutzbaren Schnittstelle unterstützen. Sie übernimmt komplexe Abläufe und kapselt diese in gut dokumentierten Funktionen. Da die Client-Library primär von weiteren Entwicklern genutzt werden soll, wird an dieser Stelle eine ausführliche Dokumentation zur Benutzung geboten.

4.7.1. Benutzung

Die Client-Library muss vor der Benutzung in der im selben Ordner befindlichen *config.ini* (Listing 8) konfiguriert werden. In Listing 7 werden die wichtigsten Funktionen der Client-Library exemplarisch ausgeführt. Die Konfigurationsdatei enthält sensible Informationen und muss durch geeignete Zugriffskontrollmechanismen geschützt werden.

⁶¹wie schon in Abschnitt 4.1 beschrieben

```

1  require_once('Path/To/Client/Client.php');
2  use Infrz\OAuth\Client\Client;
3
4  // initialize Client
5  $client = new Client();
6  // generate the Authorization Request Uri
7  $auth_request_uri = $client->getAuthorizationRequestUri();
8  // exchange auth_code for access_token
9  $access_token = $client->getAccessToken($code);
10 // retrieve user information by access_token
11 $user = $client->getUser($access_token)

```

Listing 7: Benutzung der OAuth InfrZ Client-Library

Bevor man einen Client initialisieren kann, muss man die entsprechende Klasse laden. Dies geschieht in den ersten zwei Zeilen⁶². Der Konstruktor (Zeile 5) akzeptiert optional einen Pfad zur benutzten Konfigurationsdatei in Form eines Strings, standardmäßig wird die im gleichen Verzeichnis befindliche *config.ini* geladen. Beim Autorisierungsprozess von OAuth wird ein Client vom Benutzer autorisiert. Damit dies geschehen kann, muss der Benutzer zunächst auf die richtige Seite weitergeleitet werden. Die URL der entsprechenden Seite kann mit der Methode `getAuthorizationRequestUri()`; (Zeile 7) generiert werden. Optional kann man der Methode noch eine URL angeben zu der der Benutzer nach erfolgreicher Autorisierung umgeleitet wird. Dabei ist zu beachten, dass diese URL zu dem selben Host führen muss wie die Client-Anwendung. Nach einer erfolgreichen Autorisierung wird der Benutzer, mitsamt Authorization Code, auf eine Client Seite weitergeleitet. Mit der Methode `getAuthToken($code)`; (Zeile 9) kann ein Authorization Code gegen ein Access Token getauscht werden. Das Access Token wird als ein Objekt des Typs `Infrz\OAuth\Client\Model\AuthToken` zurückgegeben. Letztendlich kann man mit der Methode `getUser($access_token)`; (Zeile 11) auf die vom Benutzer freigegebene Information zugreifen. Der Benutzer wird als ein Objekt des Typs `Infrz\OAuth\Client\Model\User` zurückgegeben.

4.7.2. Konfiguration

Listing 8 zeigt eine unveränderte Client Konfiguration. Um Unklarheiten zu vermeiden wird an dieser Stelle auf die einzelnen Einstellungsmöglichkeiten eingegangen.

⁶²von hier an beziehen sich Zeilenangaben auf Listing 7

```

1 ;### This is a basic configuration file for the oauth infrz client.
2 ;### Make sure to secure this file properly!
3
4 ; The client_id from the client which is using the service.
5 client_id = "INSERT_YOUR_CLIENT_ID_HERE"
6
7 ; The matching client_secret to the client_id.
8 client_secret = "INSERT_YOUR_CLIENT_SECRET_HERE"
9
10 ; The URL to the OAuth-Infrz server aka resource owner.
11 server_url = "https://localhost"
12
13 ; The default redirect uri to your page.
14 default_redirect_uri = "https://your.domain.com/your_service/"

```

Listing 8: config.ini der Client-Library

Jeder Client-Anwendung sind eine eindeutige **Client ID** (Zeile 5)⁶³ und ein vertrauliches **Client Secret** (Zeile 8) zugewiesen. Bei beiden handelt es sich um zufällige Zeichenketten, die vom Client-Moderator auf der *Clientseite* eingesehen werden können. Mit der **Server URL** (Zeile 11) ist die URL des OAuth InfrZ Dienstes gemeint. Aktuell läuft dieser auf „*https://svs-sso.informatik.uni-hamburg.de*“ und ist nur fachbereichsintern erreichbar. Die **Default Redirect URI** (Zeile 14) ist die standardmäßig genutzte URL zu der ein Benutzer nach erfolgreicher Autorisierung weitergeleitet wird. Dieser Endpunkt sollte mit dem *Authorization Code* umgehen können und weitere angemessene Schritte durchführen. Alle Werte sollten mit den Einstellungen des *Clients* beim OAuth InfrZ übereinstimmen.

4.7.3. Client-Server Kommunikation

Die Kommunikation zwischen Client und Server erfolgt durch HTTP-Requests. Die Antworten des Servers enthalten JSON-Codierte Objekte. Access-Tokens werden mit einem POST-Request an `/authorize/token` ausgeteilt. Ein gültiger Request muss im Message-Body die folgenden Variablen definieren:

grant_type

Der `grant_type` bestimmt ob man einen Authorization-Code oder ein Refresh Token benutzt. Akzeptiert werden die Werte `"authorization_code"` und `"refresh_token"`.

client_id

Die Client-ID des Clients, der das Access-Token anfordert.

client_secret

Das Client-Secret des Clients, der das Access-Token anfordert.

⁶³von hier an beziehen sich Zeilenangaben auf Listing 8

code

Der Authorization-Code oder das Refresh-Token, mit dem das neue Access-Token angefordert wird.

redirect_uri

Die Redirect-URI des Clients, wie in der Konfiguration angegeben.

Bei einem gültigen Request wird ein Access-Token wie in Listing 9 als Antwort zurückgegeben. Es enthält das Access-Token, sowie ein optionales Refresh-Token, das freigegebene Scope und einen Ablaufzeitpunkt. Das Scope wird als Array von Strings, der Ablaufzeitpunkt als UNIX-Timestamp⁶⁴ angegeben. Das Access-Token in Listing 9 enthält ein vollständiges Scope und ist somit eine Zugriffsberechtigung auf alle verfügbaren Ressourcen.

```
1 {  
2   "type": "AccessToken",  
3   "access_token": "yg-vlacBBSPJdAGfB-0NuaS-50x0zYlRBZHtLWzeG-ERlH",  
4   "refresh_token": "OlYonB-F0aLg9i3HXF7zrdZfVcjL4X5JWMQ4zEtg_h-13y",  
5   "scope": ["kennung", "name", "email", "groups"],  
6   "expires_at": 1366031142  
7 }
```

Listing 9: JSON-Codiertes Access Token

Der eigentliche Ressourcenzugriff auf die Benutzer-Information erfolgt mit einem GET-Request an `/user?access_token={access_token}`. Anstelle von `"{access_token}"` muss ein gültiges Access-Token angegeben werden. Die Antwort enthält ein Benutzer-Objekt mit der im Scope freigegebenen Benutzer-Information.

```
1 {  
2   "type": "User",  
3   "kennung": "2king",  
4   "name": "Joe King",  
5   "email": "joe@king.com",  
6   "groups": ["admin", "svs", "oauth_client"],  
7   "scope": ["kennung", "name", "email", "groups"]  
8 }
```

Listing 10: JSON-Codiertes User Objekt

Fehlermeldungen enthalten den Fehler-Typ, eine Fehler-Beschreibung und den HTTP-Statuscode. Fehler sollen bei der Entwicklung von Client-Anwendungen berücksichtigt werden.

⁶⁴vgl. Benjamin Sintay: Unix Timestamp. <http://www.unixtimestamp.com> (Zugriff am 15.04.2013)

```

1 {
2   "type": "Error",
3   "error": "not_found",
4   "error_description": "The requested URL was not found on this server.",
5   "http_status": 404
6 }

```

Listing 11: JSON-Codierte Fehlermeldung

4.8. Ausblick

4.8.1. Client-Library

Die **Client-Library** ist in PHP geschrieben und kann somit nur in PHP geschriebenen Webanwendungen verwendet werden. Nicht jede Webanwendung ist in PHP geschrieben und nicht jeder Entwickler schreibt seine Webanwendung bevorzugt in PHP. Aus diesem Grund ist es sinnvoll die Client-Library in weiteren Programmiersprachen anzubieten. Aufgrund ihrer Verbreitung bietet es sich an die Client-Library in Ruby, Python und Java zu übersetzen.

Damit die Client-Library in einer Programmiersprache umgesetzt werden kann muss die Programmiersprache *TLS-Requests* auswerten können. Wenn ein *JSON-Parser* oder eine entsprechende Library vorhanden ist, kann eine Client-Library mit geringem Aufwand geschrieben werden. Die drei oben genannten Programmiersprachen erfüllen beide Kriterien. Ferner ist davon auszugehen, dass Programmiersprachen in denen Webanwendungen geschrieben werden können, auch alle notwendigen Kriterien erfüllen.

4.8.2. Host

Es ist sinnvoll die Kommunikation zwischen der Client-Anwendung und OAuth InfRZ auf die Hosts⁶⁵ der Client Anwendung zu begrenzen. Ein solcher whitelist-Ansatz sollte sinnvollerweise vom Client-Moderator verwaltet werden. Diese Maßnahme erschwert Angreifern den unbefugten Zugriff. Eine solche Begrenzung könnte über Host- und IP-Adressen geschehen. Eine zusätzliche Authentifizierung des Clients durch ein TLS-Zertifikat würde für weitere Sicherheit sorgen. Ein Angreifer müsste zusätzlich zu Client-Secret, Host-Adresse und Kontrolle über die Kommunikation auch das TLS-Zertifikat des Clients besitzen. Die Datenbank und Benutzeroberfläche von OAuth InfRZ sind für die Implementierung einer whitelist vorbereitet.

⁶⁵dabei gilt es nur die Hosts zu erlauben die mit OAuth InfRZ kommunizieren

4.8.3. Client-Moderator

OAuth InfRZ hat über das Active Directory Zugriff auf die Gruppen eines Benutzers. Daher bietet es sich an, Client-Moderatoren durch die Zugehörigkeit zu einer dedizierten Gruppe im Active Directory zu autorisieren. Dadurch wird die Verwaltung von zusätzlichen Benutzerkonten für Client-Moderatoren hinfällig, da diese ihren Informatik Rechenzentrum Account nutzen können. Durch eine solche Implementation ist die Verwaltung der Client-Moderatoren Aufgabe des Rechenzentrums. OAuth InfRZ ist auf die Verwaltung der Client-Moderatoren durch Gruppenzugehörigkeit vorbereitet.

In der aktuellen Implementation von OAuth InfRZ ist jeder Client-Anwendung genau *ein* Client-Moderator zugewiesen. Das ist für den Universitätsbetrieb unpraktisch. Eine Erweiterung auf eine Gruppe von Benutzern ist möglich und sollte in Betracht gezogen werden.

Literatur

- [AE12] Axel Arnbak, Nico Van Eijk: Certificate Authority Collapse: Regulating Systemic Vulnerabilities in the HTTPS Value Chain. 40th Research Conference on Communication, Information and Internet Policy, 2012 TRPC, Arlington (US-VA) September 2012
- [AK12] George Argyros, Aggelos Kiayias: I Forgot Your Password: Randomness Attacks Against PHP Applications. USENIX Security '12, Bellevue (US-WA) August 2012.
- [Boy12] Ryan Boyd: Getting Started with OAuth 2.0. First Edition, O'Reilly, Sebastopol (US-CA) 2011.
- [Cro06] Douglas Crockford: JSON: The Fat-Free Alternative to XML. XML 2006, Boston (US-MA) Dezember 2006.
- [DR11] Thai Duong, Julianio Rizzo: Here Come The \oplus Ninjas. ekoparty Security Conference 9^o edition, Buenos Aires, Argentinien, Mai 2011.
- [Fie00] Roy T. Fielding: Architectural Styles and the Design of Network-based Software Architectures. University Of California, Dissertation, Irvine (US-CA) 2000.
- [Hur97] Jani Hursti: Single Sign-On. Seminar on Network Security, Helsinki University of Technology, Helsinki, Finnland 1997.
- [Kay93] Alan C. Kay: The Early History Of Smalltalk. History of programming languages II (511-598). Association for Computing Machinery, New York (US-NY) 1993.
- [KH10] Andreas M. Kaplan, Michael Haenlein: Users of the world, unite! The challenges and opportunities of Social Media. Business Horizons 53/1 (2010) 60-68.
- [Leb11] Jonathan LeBlanc: Programming Social Applications. First Edition, O'Reilly, Sebastopol (US-CA) 2011.
- [Po09] John Policelli: Active Directory Domain Services 2008 How-To. First Edition, Sams Publishing, Indianapolis (US-IN) 2009.
- [Ree08] Will Reese: Nginx: the High-Performance Web Server and Reverse Proxy. Linux Journal 173 (2008) Article 2.
- [RFC1866] Daniel W. Connolly, Tim Berners-Lee: Hypertext Markup Language - 2.0. <http://tools.ietf.org/html/rfc1866> (Zugriff am 23.02.2012)
- [RFC2251] Mark Wahl, Tim Howes, Steve Kille: Lightweight Directory Access Protocol (v3). <http://tools.ietf.org/html/rfc2251> (Zugriff am 02.03.2013))

- [RFC5246] Kurt D. Zeilenga: Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. <http://tools.ietf.org/html/rfc4510> (Zugriff am 27.12.2012)
- [RFC5246] Eric Rescorla, Tim Dierks: The Transport Layer Security (TLS) Protocol - Version 1.2. <http://tools.ietf.org/html/rfc5246> (Zugriff am 27.12.2012).
- [RFC5849] Eran Hammer-Lahav: The OAuth 1.0 Protocol. <http://tools.ietf.org/html/rfc5849> (Zugriff am 25.12.2012).
- [RFC6749] Dick Hardt: The OAuth 2.0 Authorization Framework. <http://tools.ietf.org/html/rfc6749> (Zugriff am 25.12.2012).
- [RFC6750] Dick Hardt, Michael B. Jones: The OAuth 2.0 Authorization Framework: Bearer Token Usage. <http://tools.ietf.org/html/rfc6750> (Zugriff am 25.12.2012).
- [Rit12] Tom Ritter: New Standards for Browser-Based Trust - The Recent Acceleration of Improvements. iSEC Partners, San Francisco (US-CA) 2012.
- [Sin12] Brijendra Singh: Network Security & Management. Third Edition, PHI Learning, Delhi, India 2012.
- [Som10] Ian Sommerville: Software Engineering. Ninth Edition, Pearson, Boston (US-MA) 2010.
- [SQL1992] Database Language SQL. Digital Equipment Corporation, Maynard (US-MA) 1992.
- [SS12] Christopher Soghoian, Sid Stamm: Certified Lies: Detecting and Defeating Government Interception Attacks Against SSL. 15th International Conference Financial Cryptography and Data Security '11, Gros Islet, St. Lucia, Februar 2011.
- [TW10] Andrew S. Tanenbaum, David J. Wetherall: Computer Networks. Fifth Edition, Pearson, Boston (US-MA) 2010.
- [WK06] Martin Wind, Detlef Kröger (Hrsg.): Handbuch IT in der Verwaltung. Auflage 2006, Springer-Verlag, Berlin (DE-BE) 2006.

Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen – benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin mit der Einstellung der Arbeit in den Bestand der Bibliothek des Departments Informatik einverstanden.

Hamburg, den 20. Mai 2013

Senad Ličina