Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Alexander Simons

Konzeption und Realisierung einer Google Chrome Extension zur Anonymisierung in sozialen Medien

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexander Simons

Konzeption und Realisierung einer Google Chrome Extension zur Anonymisierung in sozialen Medien

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr.-Ing. Olaf Zukunft
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Eingereicht am: 25. Juli 2014

Alexander Simons

Thema der Arbeit
Konzeption und Realisierung einer Google Chrome Extension zur Anonymisierung in sozialen Medien

Stichworte
Anonymisierung, Soziale Medien, Google Chrome Extension, Namensgenerator

Kurzzusammenfassung
Diese Bachelorarbeit beschäftigt sich mit der Entwicklung einer Client-Komponente, in Form einer Google Chrome Extension, mit welcher es möglich sein wird Benutzernamen in Regestrierungsformulare von Social Media Seiten einzufügen. Dabei werden die Namen mittels einer Server-Komponente generiert, welche einen REST-Service und eine Datenbasis anbietet. Es wird möglich sein, die die Client-Komponente zu konfigurieren, so dass unterschiedliche Komplexitätsstufen definiert werden können. Dabei reichen diese von einfachen, sprechenden Namen bis hin zu Namen, die eine maximale Entropie aufzeigen. Mit diesem Verfahren ist es möglich die Mustererkennung von MOBIUS zu umgehen.
Alexander Simons

Title of the paper
Concept and implementation of a Google-Chrome extension for anonymized namegeneration in social media

Keywords
Anonymization, Social Media, Google Chrome Extension, Namegenerator

Abstract
This bachelor thesis is about the realization of a client-component, in terms of a google chrome extension. By the extension it will be possible to generate usernames and insert them into registration forms of social media websites. The names will be generated by a server componenten which offers a REST-Service and a database. It will be possible to configure the complexity of the usernames on client side in a range from simple, speaking usernames to usernames with a maximal entropy. With that method it will be possible to bypass the recognizing patterns of MOBIUS.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1. Einführung

Die letzten Jahre haben zunehmend gezeigt, dass ein Interesse daran besteht Internetbenutzer zu identifizieren und deren Daten zu sammeln. Insbesondere Gewohnheiten, wie sich die Benutzer im Internet bewegen, welche Seiten aufgerufen werden und weiteres Surfverhalten stehen dabei im Fokus. Wirtschaftliche Interessen werden deutlich, wenn sich Geschäftsmodelle entwickeln, die mit den gesammelten Daten und Profilen Profit betreiben. Dies steht im klaren Widerspruch dazu, dass persönliche Daten nicht ohne Genehmigung weitergegeben dürfen.

Auch Regierungen stehen zunehmend im Fokus an den Daten und daraus entstehenden Profilen von Internetbenutzern Interesse zu haben. Mit der Begründung fehlender Kontrollmechanismen, Terrorismusbekämpfung und dem Nutzen, den das Datensammeln haben soll, werden Kontrollapparate und Datenanalysezentren aufgebaut mit deren Hilfe eine *frühzeitige* Erkennung von Bedrohungen erzielt werden soll, welche im klaren Kontrast zu existierenden Bürger -und Menschenrechten stehen.

Diese Arbeit beschäftigt sich mit einem Verfahren, welches Mustererkennungen bzgl. der Vergabe von Benutzernamen einsetzt um Benutzern über verschiedene Social Media Webseiten zu identifzieren. Es soll gezeigt werden, dass sich mit vertretbarem Aufwand Benutzernamen generieren lassen, die diese Muster verschleiern und somit die Identifzierung erschweren.

1.1. Zielsetzung

Ziel dieser Arbeit ist die Konzeptionierung und Realisierung eines Anonymisierungs-Tools, bestehend aus Server- und Client- Komponente, um die Mustererkennung aus der Arbeit “Connecting Users across Social Media Sites: A Behavioral-Modeling Approach” von Reza Zafarani und Huan Liu an der Computer Science and Engineering Arizona State University zu umgehen. Dabei wird die Client-Komponente als Google-Chrome Extension realisiert. Der Wiedererkennungswert soll durch sprechende, aber nicht ähnliche Namensmuster verschleiert werden, da sich die MOBIUS-Software diese zunutze macht, um Benutzer über verschiedene Social-Media Seiten hinweg zu identifizieren.

1.2. Aufbau der Arbeit

Gegliedert ist diese Arbeit in folgende fünf Abschnitte:

- Zu Beginn wird auf den Begriff der Anynomisierung in sozialen Medien eingegangen. Dann werden die Grundlagen der eingesetzten Technologien auf Server -und Clientseite erläutert, sowie ein Bezug zu aktuellen Geschehnissen hergestellt.
- Anschließend werden die Ergebnisse des MOBIUS-Projekts vorgestellt und ein Konzept um dessen Mustererkennung zu umgehen.
- Im Kapitel “Client-Komponente” wird auf die Implementierung der Google-Chrome Erweiterung eingegangen.
- Das Kapitel “Server-Komponente” beschreibt die Entwicklung der Server-Komponente dieser Arbeit.
- Der letzte Abschnitt “Bewertung” resümiert die wesentlichen Ergebnisse des entwickelten Konzepts.

Eine Beschreibung des Inhalts der beiliegenden CD ist im Anhang A zu finden.

2. Grundlagen

Diese Kapitel beschreibt im ersten Abschnitt den Begriff der Anonymisierung in sozialen Medien und wie Handel mit Daten betrieben wird. Im zweiten Abschnitt folgen die Technologien, welche zur Entwicklung der Client-Komponente in Form einer Google Chrome Extension [10] benötigt werden. Darüber hinaus werden jene Techniken vorgestellt mit denen es möglich ist eine Extension mittels Test-Driven Development [1] (TDD) zu entwickeln. Das dritte Kapitel umfasst die Technologien, die zur Realisierung der Server-Komponente benötigt werden. Abschließend wird ein aktueller Bezug zu derzeitigen Geschehnissen hergestellt, der verdeutlich, wie mit (Meta-) Daten Profile erzeugt werden.

2.1. Anonymisierung in sozialen Medien

Soziale Medien ist ein Sammelbegriff für im internet-basierte mediale Angebote, die aus dem sogenannten Web 2.0 hervorgegangen sind. Die soziale Interaktion, in Form von Kommunikation und dem Austausch von nutzergenerierteren Inhalten (sog. User-Generated Content), stehen dabei im Vordergrund. Durch die vernetzte Struktur in sozialen Medien entsteht zunehmend auch eine kommerzielle Bedeutung, da so werbewirksame Inhalte und Nachrichten schneller an den potentiellen Konsumenten herangetragen werden können. [26].

Der Gesetzgeber unterscheidet prinzipiell zwischen Anonymisierung und Pseudonymisierung [5]:

- Unter Verändern der personenbezogengen Daten, so dass diese Daten nicht mehr einer Person zugeordnet werden können, versteht man Anonymisierung.
- Wird der Name oder ein anderes Identifikationsmerkmal durch ein Pseudonym (zumeist eine mehrstellige Buchstaben- oder Zahlenkombination, auch Code genannt) ersetzt, um die Identifizierung des Betroffenen auszuschließen oder wesentlich zu erschweren, bezeichnet man dies als Pseudonymisierung.

Im Gegensatz zur formalen Definition des Gesetzgebers ist die Unterscheidung im alltäglichen Gebrauch eher selten und eine Vermischung der beiden Begriffe ist üblich.

2.1.1. Handel mit Daten

Um Käufergruppen besser und effizienter ansprechen zu können hat sich mittlerweile ein Markt für Benutzerprofile entwickelt. Als Beispiel für kommerzielle Ansätze sei Google-Mail genannt, das für den Benutzer zwar kostenlos ist, dafür aber automatisiert alle E-Mails mitliest, um personalisierte Werbeanzeigen zu schalten. Ein ähnliches Model betreibt Facebook um passende Werbung anzuzeigen. Dazu werden aus den vorhandenen Benutzerdaten Informationen zum Kaufverhalten abgeleitet. Grundsätzlich gilt: je mehr Daten über eine Person vorhanden sind desto besser kann die Werbung auf diese zugeschnitten werden. Dies versuchen sich viele Unternehmen und Geschäftsmodelle zu Nutze zu machen.

2.1.2. Bedeutung für die Wirtschaft

Unternehmen wie Facebook, bei dem zum Zeitpunkt des Börsengangs eine Bewertung von 100 Milliarden Dollar im Raum stand [13], sind auf große Datenmengen angewiesen. Ein Geschäft, das sich vor allem im Bereich der Werbung für Facebook zu lohnen scheint. Die Werte des Jahresumsatzes von 7,9 Milliarden Dollar und einem Gewinn von 1,5 Miliarden Dollar sprechen eine deutliche Sprache [29].

2.2. Client-Komponente

Für die Erstellung einer Google Chrome Extension wird üblicherweise HTML5, CSS3 und JavaScript benötigt, so dass sie im Kern eine HTML5-Anwendung darstellt [11]. Hinzu kommt eine sogenannte Manifest-Datei im JSON-Format mit welcher das Plugin konfiguriert wird. Hierzu zählen die Berechtigungen des Plugins, sowie die verwendeten HTML und JavaScript-Dateien.

Die Client-Komponente kommuniziert mit dem Server mittels AJAX-Aufrufen und erkennt die jeweiligen Registrierungsseiten. Bei Aufruf einer solchen Seite wird dem Benutzer ein Auswahl-Menü angezeigt in dem er weitere Aktionen wählen kann. Bei allen anderen Seiten wird dieses Menü ausgeblendet. Der vollständige Funktionsumfang der Komponente wird in Kapitel 4 behandelt.

2.2.1. Anforderungen an die Client-Komponente

Bei der Entwicklung einer Google Chrome Extension wird das Test-Driven Development nicht standardmäßig angeboten respektive unterstützt. Da dies aber für die Realisierung der

einzelnen Tests im weiteren Verlauf dieser Arbeit notwendig ist, entstehen eine Reihe von Anforderungen an die eingesetzten Technologien:

- Möglichkeit des Test-Driven Developments
- Verbreitung
- Komfortable Manipulation von HTML5-Elementen
- Kommunikation mit Server-Komponente
- Dokumentation
- Integration in eine Google Chrome Chrome Extension

Auf die vollständige Liste der Anforderungen wird in Kapitel 4.1 eingegangen.

2.2.2. Verwendete Technologien für die Client-Komponente

Durch den Einsatz der weit verbreiteten und beliebten JQuery [24] JavaScript-Library ist eine komfortable Manipulation der HTML-Objekte auf den jeweiligen Registrierungsseiten der Social-Media Webseiten möglich. Hierbei sind insbesondere die Formularfelder (Input-Fields) von Interesse. Darüber hinaus bietet sie eine umfangreiche AJAX-API an, welches das Test-Driven Development sowie die Kommunikation zum Server vereinfacht. JavaScript-Frameworks wie dojo-Toolkit [7] und Sencha [27] verfügen über ähnliche Ansätze wie JQuery, sind aber weniger verbreitet und im Vergleich eher schwergewichtig, weshalb im Folgenden JQuery eingesetzt wird.

Um ein Test-Driven Development [1] für die JavaScript Module durch Unit-Tests zu ermöglichen, wurde das im JavaScript-Umfeld bekannte QUnit [23] benutzt, welches im Rahmen der Entwicklung von JQuery entstanden ist. Test-Frameworks wie Jasmine [22] bieten ein sehr ähnliches Spektrum an Funktionialität, basieren aber auf einem anderen Konzept. Während Jasmine einen Behaviour Driven Development Ansatz verfolgt, ist es bei QUnit ein Ansatz des Test Driven Development. Bei der Entscheidung des Frameworks war insbesondere die einfache Integration mittels JQuery ausschlaggebend. Auf Mocking-Frameworks wie beispielsweise Sinon.JS [16] wurde absichtlich verzichtet, da diese keinen Mehrwert für diese Arbeit darstellen.

Die entwickelte Extension ist unter der Google Chrome Version 32.0.1700.107 entwickelt worden, dabei kamen JQuery in der Version 2.1.0, sowie QUnit in der Version 1.14.0 zum Einsatz.

2.3. Server-Komponente

Die Server-Komponente bietet die Schnittstellen zur Namensgenerierung für die Client-Komponente an. Außerdem enthält sie den Algorithmus zur Generierung der Namen sowie die Namensdatenbank. Auf den vollständigen Funktionsumfang sowie die Implementierung wird im 5. Kapitel eingegangen.

2.3.1. Anforderungen an die Server-Komponente

Die wichtigsten Anforderungen an die in dieser Arbeit zu entwickelnden Server-Komponente lauten:

- REST-Service
- Skalierbarkeit
- Datenbankintegration
- Verfügbarkeit
- Dokumentation
- Installationsroutinen
- Möglichkeit der testgetriebenen Entwicklung
- Open Source

Eine detaillierte Übersicht aller Anforderungen befindet sich unter 5.1.

2.3.2. Verwendete Technologien für die Server-Komponente

Die Anbindung von Web-Services an Datenbanken ist gängige Praxis. Die Technologie-Landschaft bietet ein breites Spektrum an proprietären, open-source oder gemischten Lösungen an. Die bekanntesten Vertreter sind:

- Microsoft .NET Technologien: MS SQL Server als (relationale) Datenbank und C# oder VisualBasic als Programmiersprache, seltener aber immer populärer auch F#. Hinzu kommt der Internet Information Services (kurz IIS) als Dienstplattform. In diesem Kontext wird üblicherweise Windows Server als Betriebssystem eingesetzt.

- Java-Technologien: im Java Enterprise Edition (Java EE) Umfeld sind Tomcat als open source Webserver respektive Webcontainer gebräuchlich. Hinzu kommen open source Datenbanken wie mySQL oder, im Zuge der Big Data Bewegung, MongoDB oder Cassandra.

- Reine open source Lösungen: Im Gegensatz zu Java, in dessen Umfeld durchaus auch proprietäre Software-Komponenten existieren, haben sich auch eine ganze Reihe von reinen open source Lösungen auf dem Markt etabliert und erfreuen sich großer Beliebtheit. Viele dieser Lösungen nutzen Apache als Webserver und einer open source Programmiersprache, wie beispielsweise Ruby, Python oder PHP. Häufig kommt dabei mySQL, PostgreSQL oder eine andere frei verfügbare Datenbank zum Einsatz.

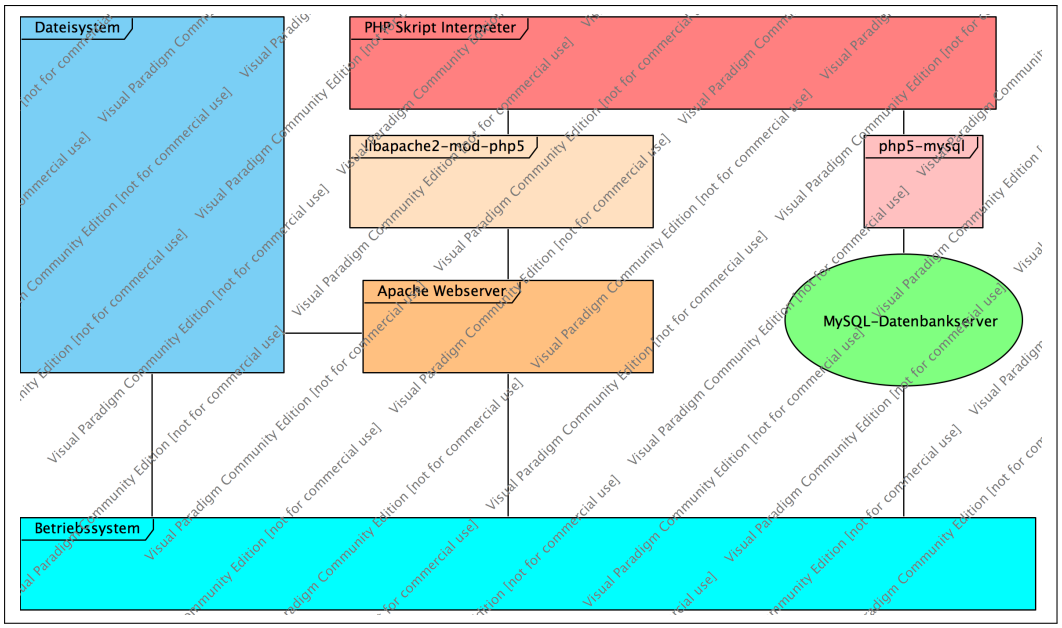

Abbildung 2.1.: LAMP Architektur

Oblgeich die Microsoft- und auch Java-Produkte sich insbesondere im Geschäftsumfeld großer Beliebtheit erfreuen, wurde diese Arbeit unter Zuhilfenahme des bekannten LAMP [17] Software-Pakets verfasst. Die Gründe hierfür sind zum einen die vollständige Einsicht des Quellcodes, die Minimierung der Kosten sowie die große Verbreitung und somit Verfügbarkeit. Schlussendlich ist es, aufgrund der mittlerweile einfachen Installationsroutinen, nahezu jedermann möglich ohne Software-Kosten die nötigen Software-Komponenten zu installieren und zu konfigurieren.

Im Kern handelt es sich bei der Serverkomponente um eine dynamische Webseite auf PHP-Basis [21], die das relationale Datenbankverwaltungssystem MySQL [19] einbindet, in welchem Namenslisten und weitere Daten gehalten werden. Um die Anforderungen an eine REST-ähnliche Applikation zu erfüllen, das Routing zu vereinfachen und den Entwicklungsaufwand zu minimieren, setzt die Komponente auf dem Slimframework [18] auf. Ferner wird der OR-Mapper des Doctrine Projects[28] für eine bessere Skalierbarkeit und Abstraktion der Datenschicht eingesetzt. Zur testgetriebenen Entwicklung der Server-Komponente wird PHPUnit [3], eine Portierung des bekannten JUnit-Test Frameworks aus der JAVA-Welt, verwendet.

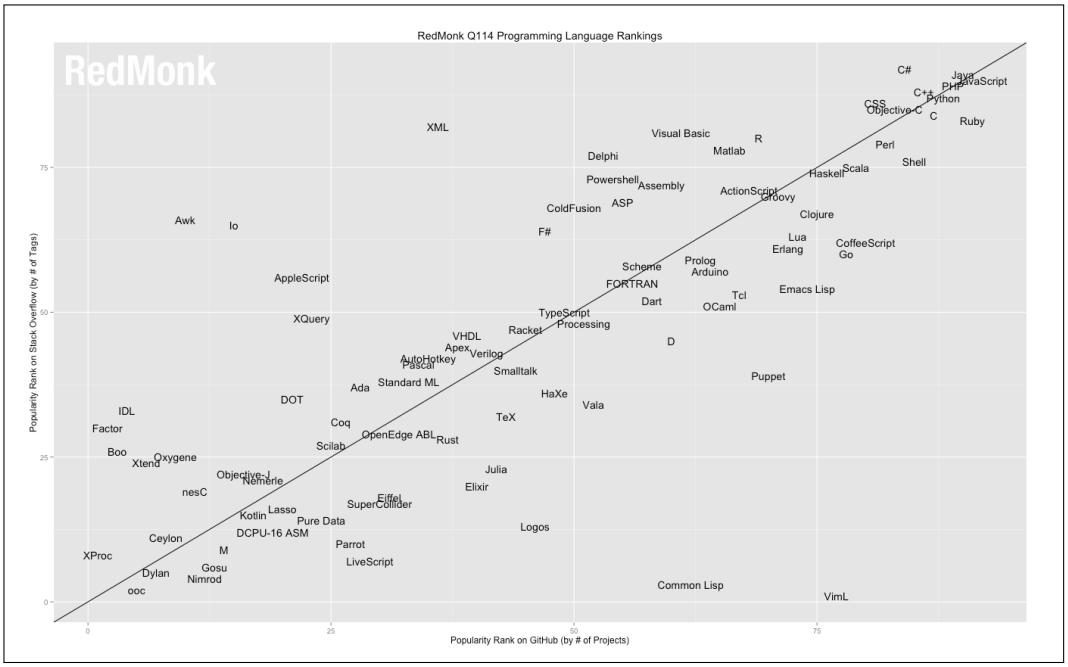

Abbildung 2.2.: RedMonk Programming Language Rankings

Quelle: http://redmonk.com/sogrady/2014/01/22/language-rankings-1-14/

Die Server-Komponenten benötigt PHP ab Version 5.3 und MySQL ab Version 5.1.

2.4. Aktueller Bezug

Wie spätestens ab 2013 durch die NSA-Spionageaffäre deutlich wird, sind es aber nicht nur kommerzielle Unternehmen, die gezielt versuchen Persönlichkeitsprofile anhand von extrem großen Datenmengen (Big Data) zu erstellen, sondern auch Regierungen und insbesondere deren Geheimdienste. Daraus ergibt sich die Frage, inwieweit man sich als Einzelperson vor derartigen Datenerhebungen schützen kann. Dies ist als klare Abgrenzung dieser Arbeit zu sehen, da der Umfang technisch, politisch wie auch soziologisch zu umfassend wäre.

2.4.1. Zusammenfassung bisheriger Ereignisse der NSA-Affäre

Im Juni 2014 begannen der Guardian und die Washington Post mit den ersten Enthüllungen rund um das Thema der Kommunikations-Überwachung. Dabei wurde zum ersten Mal deutlich, welche Ausmaße diese Überwachung durch die US-amerikanische National Security Agency (NSA) und andere westliche Geheimdienste annimmt. Durch den Whistleblower und ehemaligen NSA-Analysten Edward Snowden, der dem Guardian vertrauliche Unterlagen überreichte, wurde das Ausmaß nach und nach immer deutlicher. Zwischenzeitlich hatte die Bundesrepuklik Deutschland die „Affäre„ als beendet erklärt; mittlerweile ist klar, dass Edward Snowden vor dem Europäischen Parlament zur NSA-Affäre befragt wird. Es dauerte bis Oktober 2013 ehe die Brisanz, durch das Aufdecken, der Tatsache, dass das Mobiltelefon von Bundeskanzlerin Angela Merkel ebenfalls abgehört wurde, auch der breiten Bevölkerung und den Medien allmählich klar wurde. Ob, und wann dies in der Vergangenheit der Falle war, wollte das Weiße Haus explizit nicht kommentieren. Um sein Missfallen auszudrücken, bestellte Guido Westerwelle in diesem Zuge den US-Botschafter ein. Es bedurfte aber zusätzlich eines weiteren Spiegel-Artikels, welcher ein Umdenken der Bundesregierung zur Folge hatte. Dieser legte dar, dass ein großflächiges Anti-Spionage-Abkommen zwischen Deutschland und den Vereinigten Staaten, respektive den sogenannten Five Eyes, einer Geheimdienstallianz bestehend aus USA, Großbritannien, Kanada, Neuseeland und Australien Bestand hat. Zuvor war bereits bekannt geworden, dass bereits millionenfach Kommunikationsdaten durch die NSA in Frankreich, Spanien und Italien monatlich abgegriffen wurden. Kurz darauf wurde öffentlich, dass die NSA ebenfalls die unverschlüsselte Kommunikation zwischen den Rechenzentren von Unternehmen wie Google und Yahoo mithört und weiter verwertet. Die Entrüstung der Internetkonzeren war dementsprechend groß und der Imageschaden schwer wieder gut zu machen. Bis dato war bereits bekannt, dass die NSA mit geheimen richterlichen Beschlüssen die Datenherausgabe von US-Unternehmen verlangt oder alternativ die Unternehmen für diese bezahlt. Anfang September 2013 berichteten der Guardian und die New York Times

über Versuche von NSA und GCHQ die verschiedenen Verschlüsselungen im Internet zu brechen oder zu umgehen. Beispielsweise wird dabei in Geräte eingedrungen um dort direkt die unverschlüsselte Kommunikation mitzuschneiden. Um in die Geräte einzudringen werden auf verschiedenste Weise Sicherheitsschlüssel organisiert, Hersteller veranlasst Hintertüren in Soft- und Hardware zu implementieren, auf dem Schwarzmarkt Hacker-Werkzeuge oder Zero-Day-Exploits erworben oder anderweitige Sicherheitslücken ausgenutzt. Ende September zeigte sich dann, aus welchem Grund die NSA derart große Datenmengen sammelt. Laut der New York Times ist es so der NSA möglich die sozialen Beziehungen überwachter Personen darzustellen. Ergänzend können hierzu Verbindungsdaten aus anderen Quellen hinzugezogen werden, Einschränkungen gibt es hierbei nicht. Dies verdeutlicht, wie auch harmlose Verbindungsdaten beziehungweise Metadaten in Summe zur Informationsgewinnung verwendet werden können. Letztlich verdeutlicht dies aber auch, dass bei sorgsam implementierter Kryptografie in Kombination mit hinreichend langen Passwörtern und/oder Schlüsseln ein gewisses Sicherheitsmaß gewahrt werden kann. [15]

3. Konzept

Im folgenden Abschnitt wird die wissenschaftliche Arbeit *Connecting Users across Social Media Sites* [32] vorgestellt. Dabei wird zunächst auf Muster aufgrund menschlicher Einschränkungen eingegangen. Anschließend werden sowohl die exogenen, wie auch die endogenen Faktoren menschlichen Handelns betrachtet, aus denen man Merkmale einer Person und somit deren Wiedererkennungswert entwickeln kann. Dann wird die Analyse respektive das Experiment der Arbeit sowie deren Ergebnisse präsentiert.

3.1. Erläuterung der Funktionsweise von MOBIUS

Einzelpersonen nutzen oft die gleichen Verhaltensmuster um Benutzernamen auszuwählen. Durch den Wiedererkennungswert dieser Muster lassen sich Personen über verschiedene soziale Medien identifzieren.

Diesem Wiedererkennungswert ließe sich vorbeugen, wenn auf den unterschiedlichen Seiten vollständig unterschiedliche Benutzernamen gewählt werden. Dabei ist wichtig, dass sich keinerlei Informationen anhand eines Benutzernamens gewinnen lassen und somit Rückschlüsse auf einen bereits existenten Namen möglich sind. Um derart unabhängige Benutzernamen zu generieren ist es theoretisch möglich Namen mit einer maximalen Entropie zu erstellen. Beispielsweise in dem man die maximalen Zeichenlänge eines Benutzernamens, der auf der jeweiligen Seite erlaubt ist, ausschöpft und einen zufällig generierten String, mit allen auf der Seite erlaubten Zeichen, erstellt. Dabei ist wichtig, dass in diesem String keine Wiederholungen auftreten. Somit hätte man einen vollständig zufallsgenerierten Benutzernamen.

Diese Anforderungen stehen allerdings im Gegensatz zu den menschlichen Fähigkeiten, da man sich derartige Zeichenketten schlecht merken kann. Menschen neigen dazu sich kurze, nicht zufällige und in sich wiederholende Zeichenketten besser merken zu können. Diese Eigenschaft kann dazu beitragen eine Identifizierungs-Funktion herzuleiten. In der MOBIUS-Studie wird so versucht anhand konsistenter Verhaltensmuster Benutzer über die Auswahl ihrer Benutzernamen zu identifizieren. Dabei werden die Muster in drei Kategorien unterteilt:

- Muster aufgrund menschlicher Einschränkungen (Patterns due to Human Limitations)

- Exogene Faktoren (Exogenous Factors)
- Endogene Faktoren (Endogenous Factors)

Die Merkmale, um die Informationen, welche aus den vorigen drei Kategorien entstehen, zu erhalten, werden dabei in drei weiter Kategorien unterteilt:

- Benutzernamen-Merkmale (Kandidat) (Username Features): werden direkt aus dem Benutzernamen generiert, beispielsweise die Länge.
- Vorausgehende Benutzernamen-Merkmale (Prior-Usernames Features): diese Merkmale beschreiben alle in der Vergangenheit benutzten Namen einer Person, z.B. die Anzahl der vorigen Benutzernamen.
- Benutzername ↔ vorausgehende Benutzernamen-Merkmale (Username ↔ Prior-Usernames Features): diese Merkmale beschreiben die Relation zwischen dem Benutzernamen (Kandidat) und vorigen Benutzernamen, bspw. deren Ähnlichkeit.

Im Folgenden werden die einzelnen Muster im Detail beschrieben.

3.1.1. Muster aufgrund menschlicher Einschränkungen

Menschen besitzen allgemein Limitierungen bezüglich Zeit und Gedächtnis, sowie Wissen. Dies beeinflusst die Neigungen, die bei der Wahl eines Benutzernamens eine Rolle spielen.

Einschränkung bezüglich Zeit und Gedächtnis

Dem Umstand Folge tragend, dass Menschen bezüglich Zeit und Gedächtnis begrenzt Kapazitäten haben, wird oft der selbe Nutzername für unterschiedliche soziale Medien verwendet. So ergibt sich ein wiederkehrendes Merkmal bzw. ein vorausgehendes Benutzernamen-Merkmal. Ein weiteres Resultat dieser Limitierungen ist die Wahrscheinlichkeit, dass ein gewählter Benutzername eine ähnliche oder gleiche Länge wie ein voriger Benutzername besitzt. Außerdem zeigt sich, dass Benutzer dazu neigen, ähnliche Namen zu erzeugen, da das Ausdenken von immer neuen Namen als unnötiger Aufwand gesehen wird.

Wissens-Einschränkungen

Unser Vokabular in jeder Sprache ist begrenzt. Üblicherweise ist dabei das Vokabular in der Muttersprache weiter gefasst, als das in einer gelernten Fremdsprache. Somit kann man die Vokabulargröße einer Sprache eines Menschen als Merkmal zur Identifizierung betrachten.

Dabei betrachtet man die zerlegbare Untermenge einer Zeichenkette und nutzt diese als Merkmal. Das so entstehende Wörterbuch kann als Basis zur Identifizierung dienen. Die Wahl der Buchstaben eines Benutzernamens obliegt üblicherweise der Sprache. Während im Chinesischen der Buchstabe *x* häufig in Namen vorkommt, so ist dies im Arabischen eher unwahrscheinlich. Somit kann die Anzahl der vorkommenden Buchstaben ebenfalls als Merkmal für vorige Benutzernamen als auch Kandidaten-Benutzernamen ausgemacht werden.

3.1.2. Exogene Faktoren

Bei Exogenene Faktoren handelt es sich um Verhaltensmuster, die aus kulturellen Einflüssen und der Umgebung, in der sich eine Person befindet, ableitbar sind.

Schreibmuster

Kulturbezogen lassen sich Rückschlüsse auf die Tastatur und auf Zufallsnamen ziehen. Beispielsweise ist *qwer1234* ein häufig genutztes Passwort für Nutzer einer QWERTY-Tastaturbelegung (US-amerikanisch), während *aeusnth* ein oft genutztes Passwort für Nutzer einer AZERTY-Tastaturbelegung (französisch) ist. Um die tastaturbezogenen Gleichmäßigkeiten zu erfassen werden weitere 15 Merkmale für jede der fünf Tastaturbelegungen festgelegt:

1. (1 Merkmal) Der Prozentsatz der Tasten, die mit der selben Hand getippt werden wie die vorige Taste. Je höher dieser Wert ist, desto weniger muss der Nutzer die Hand zum Tippen wechseln.
2. (1 Merkmal) Der Prozentsatz der Tasten, die mit dem selben Finger, wie die vorige Taste getippt werden.
3. (8 Merkmale) Der Prozentsatz der Tasten, die mit jedem Finger getippt werden. Daumen werden dabei nicht berücksichtigt.
4. (4 Merkmale) Der Prozentsatz der Tasten, die in Reihe getippt werden: obere Reihe, untere Reihe, Zahlenreihe, Reihe der “Home“-Tasten.
5. (1 Merkmal) Die Durchschnittliche Entfernung (in Metern), die beim Tippen eines Benutzernamens zurückgelegt wird.

Diese Merkmale werden bei den Kandidaten-Benutzernamen und den vorigen Benutzernamen berücksichtigt.

Sprachmuster

Anhand einer Wortdatenbank mit 40 Millionen Wörtern pro Sprache (21 europäische Sprachen) werden Rückschlüsse auf die Sprache einer Person gezogen, welches als zusätzliches Merkmal definiert wird. Die Spracherkennung wird dabei auf vorige Benutzernamen angewendet. Dabei wird es sich zunutze gemacht, dass Benutzer oft auf der selben Sprache oder aus der selben Menge von Sprachen Benutzernamen wählen.

3.1.3. Endogene Faktoren

Endogene Faktoren, also Faktoren die aus der Persönlichkeit einer Person entstammen, spielen eine wesentliche Rolle bei der Wahl eines Benutzernamens. Die Unterscheidung dabei findet in drei Kategorien statt. Zum einen die persönliche Attribute, wie beispielsweise Name, Geschlecht, Alter etc. zum anderen Wesenszüge sowie Gewohnheiten. Bei persönlichen Attributen und Wesenszügen ist der Übergang nahezu fließend, während die Gewohnheiten einer Person separat gesehen werden können.

Persönliche Attribute und Wesenszüge

Das zugrunde liegende Verfahren hat Schwierigkeiten bei der Wahl eines Berges oder einer Stadt als Benutzernamen. Dennoch bietet die Verteilung von Buchstaben auch hier einen Anhaltspunkt zur Spracheerkennung. Ferner können so Wörter identifiziert werden, die für die jeweilige Person von Bedeutung sind. Vollkommen zufällig ausgewählte Benutzernamen bieten keine Möglichkeit einen Wiedererkennungswert zu erlangen. Dennoch kann so das Bedürfnis nach Privatspähre ermittelt und gemessen werden. Die Entropie eines Benutzernamens, im Sinne der Verteilung des Alphabets, wird als weiteres Merkmal behandelt.

Gewohnheiten

Da alte Gewohnheiten schwer abzulegen sind, haben diese signifikante Auswirkungen auf die Erstellung von Benutzernamen. Eine der häufigsten Gewohnheiten ist die Benutzernamen-Modfizierung. Dabei werden Pre- oder Suffixe mit einem alten Benutzernamen kombiniert, alte Benutzernamen verkürzt oder Teile dessen weiterverwendet. Ebenso werden einzelne Buchstaben durch andere ersetzt. Hinzugefügte Pre- oder Suffixe werden ermittelt, in dem geprüft wird ob ein Benutzername ein Substring eines vorigen ist. Um Verkürzen zu erkennen wird ein paarweiser Vergleich der Buchstaben des Benutzernamens und den Vorgängern vorgenommen. Ausgetauschte und hinzugefügte Buchstaben werden mittels Edit (Levinshtein) und Dynamic Time Warping (DTW) gemessen und als Merkmal gespeichert.

Benutzer tendieren dazu gleiche Benutzernamen erneut zu verwenden. Dennoch ist die Ähnlichkeit mit den bisher vorgestellten Methoden relativ schwierig zu gewährleisten. Beispielsweise sind *gateman* und *nametag* sehr ähnlich nur jeweils rückwärts geschrieben. Mittels der Kullback-Lieber (KL) Divergenz lassen sich solche Veränderungen messen, allerdings ist sie nicht für metrische Vergleiche geeignet, so dass die Jensen-Shannon Divergenz genutzt wird, welche eine Messung der zugrunde liegenden KL möglich macht. Um die Überlappung der Benutzernamen zu messen wird die Jaccard Distanz als weiteres Merkmal betrachtet.

Man kann annehmen, dass die Ananderreihung der Buchstaben eines Benutzernamens auf dem vorigen Wissen, also dem vorigen Benutzernamen, beruht. Somit kann eine Schätzung eines zukünftigen Namens abgegeben werden. Dazu wird die Modified Kneser-Ney (KMN) smoothing Technique verwendet und als weiteres Merkmal verwendet.

In Summe wurden so 414 Merkmale entwickelt, mit denen bereits genutzte und neue Benutzernamen untersucht werden können.

3.1.4. Experiment

Im folgenden Abschnitt wird die MOBIUS Methodik systematisch evaluiert. Zunächst wird beschrieben, wie MOBIUS eine Identifizierungs-Funktion erlernt. Anschließend wird untersucht, wie sich bei verschiedenen lernenden Algorithmen die Merkmal-Erkennung auf die Performance auswirkt. Dann erfolgt eine Analyse der Merkmale nach Wichtigkeit und eine Untersuchung, wie die Anzahl der Benutzernamen und die Anzahl der Merkmale die Lerngeschewindigkeit beeinflussen. Bevor auf das eigentliche Experiment eingegangen wird, erfolgt zunächst ein Abschnitt darüber, wie die Daten gesammelt und aufbereitet werden.

Aufbereitung der Daten

Um Benutzernamen über verschiedene Seiten hinweg zu sammeln, wurden insbesondere Soziale Netzwerke, Blogs, Blog Werbeportale und Foren als Quellen genutzt. So konnten mehrere Identitäten des selben Benutzers gesammelt werden. Insgesamt wurden 100.179 (c-U) Paare gesammelt. Wobei c ein Benutzername und U die Menge der vorausgehenden Benutzernamen ist. c und U gehören zu der selben Person. Die Datenmenge beinhaltet Benutzernamen von 32 Internetseiten, wie beispielsweise, Flickr, Reddit, StumbleUpon, und YouTube. Die gesammelten Paare werden als positive Exemplare in der Datenmenge betrachtet. Für negative Exemplare, werden per Zufall Paare der Art (ci-Uj) erzeugt, so dass ci von einem positiven Exemplar stammt and Uj von einem anderen Exemplar (i/=j), damit sichergestellt ist, dass es sich um eine andere Person handelt. In Summe wurden so 100.179 positive und 100.179 negative Exemplare

generiert, so dass die Summe etwa bei 200.000 Exemplaren liegt. Anschließend wurden die 414 Merkmale für diese berechnet und im weitere Verlauf verwendet.

Identifizierungs Funktion

Um mit den unterschiedlichen berechneten Merkmalen arbeiten zu können, wurde mit dem Bayes-Klassifikator gearbeitet, um eine Vorhersage über die Wahrscheinlichkeit zu treffen, ob ein Benutzername zu einer Person gehört. In der angestellten Untersuchung, wurde so eine Genauigkeit von 91.38% erreicht. Zum Vergleich wurden ebenfalls die Arbeiten [31] und [20] betrachtet, welche eine Genauigkeit von 66.00% respektive 77.59% erreichten. Außerdem wurden zum weiteren Vergleich noch drei Basisstudien herangezogen

- Exakte Benutzernamen Übereinstimmung mit einer Genauigkeit von 77.00%
- Übereinstimmung eines Teils des Namens mit einer Genauigkeit von 63.12%
- Buchstabenmuster mit einer Genauigkeit von 49.25%

Das nächste Kapitel beschäftigt sich mit der Wahl des lernenen Algorithmus.

Wahl des lernenden Algorithmus

Um einen lernenden Algorithmus auszuwählen wurde eine Klassifizierung über eine Reihe von lernenden Verfahren mittels 10-facher Kreuzvalidierung durchgeführt. Die Ergebnisse der einzelnen Methoden waren dabei nicht signifikant. Dies verdeutlicht, dass wenn ausreichend Informationen in den Merkmalen vorliegen, die Benutzer-Identifizierung verhältnismäßig akkurat und nicht empfindlich gegenüber der Wahl des lernenden Algorithmus ist. Die l1-regulierte logistische Regression ist dabei die exakteste Methode, weshalb sie im Folgenden verwendet wurde. Ferner wurde versucht zu ermitteln, ob alle Merkmale notwendig sind und ob es sinnvoll ist weitere Merkmale hinzuzufügen.

Analyse der Merkmalgewichtung

Bei der Merkmalgewichtung werden die Merkmale identifiziert denen die bedeutendste Auswirkung bei der Aufgabe der Klassifizierung zukommt. Dabei wurden logistische Regressionkoeffizienten verwendet, so dass sich die Liste der zehn wichtigsten Merkmale wie folgt zusammensetzt:

1. Standardabweichung des normalisierten Bearbeitungsdistanz zwischen Benutzernamen und vorigen Benutzernamen

2. Standardabweichung des normalisierten längsten gemeinsamen Teilzeichenkette zwischen Benutzernamen und vorigen Benutzernamen
3. Beobachtung der Gleichartigkeit von Benutzernamen
4. Einzigartigkeit von vorigen Benutzernamen
5. Exakte Übereinstimmung: Anzahl der Benutzernamenkandidaten unter vorigen Benutzernamen
6. Jaccard Ähnlichkeit zwischen Alphabetverteilung von den Benutzernamen und vorigen Benutzernamen
7. Standardabweichung der Entfernung, wenn ein voriger Benutzername auf einer QWERTY-Tastatur eingegeben wird
8. Entfernung, wenn ein Benutzernamenkandidate auf einer QWERTY-Tastatur eingegben wird
9. Standardabweichung des längsten gemeinsamen Teilzeichenkette zwischen zwischen Benutzernamen und vorigen Benutzernamen
10. Median der längsten gemeinsamen Subsequenz zwischen Benutzernamen und vorigen Benutzernamen

Es zeigt sich, dass alleine mit diesen zehn Merkmalen bereits eine logistische Regression mit einer Genauigkeit von 92.72% erreicht wird. Des Weiteren hat sich gezeigt, dass Zahlen [0-9] im Durchschnitt höher gewichtet sind, als (englische) Buchstaben [a-z] und sich somit besser zum Identifizieren von Personen eignen. Außerdem wurde festgestellt, dass nicht-englische Buchstaben (z.B. Umlaute, +, oder &) sich gut zum Identifzieren von Personen eignen. Insgesamt bleibt festzustellen, dass sich diese zehn Merkmale gut für eine Identifizierung eignen.

Abnehmender Ertrag beim Hinzufügen von mehr Benutzernamen und mehr Merkmalen

Es wird oft unterstellt, dass die Identifizierung einfacher wird, wenn mehr vorige Benutzernamen einer Person bekannt sind. In dieser Untersuchung wird ein monotoner Trend ab 30 vorigen Benutzernamen nachgewiesen. Mit anderen Worten, ab 25 Benutzernamen oder mehr wird kein Mehrwert für die Identifizierung erreicht. Am schwierigsten bleibt die Erkennung,

wenn nur ein voriger Benutzernamen bekannt ist. Für gewöhnlich sind 25 Benutzernamen schwer zu ermitteln, so dass, v.a. mit Blick auf die Performance sieben Benutzernamen einen guten Wert darstellen.

Ähnlich gestaltet es sich beim Hinzufügen von mehr Merkmalen. Auch hier bieten mehr Merkmale keinen Mehrwert. Es bleibt festzuhalten, dass das Hinzufügen von mehr bekannten vorigen Benutzernamen einen größeren Nutzen bringt als das Hinzufügen von weiteren Merkmalen.

3.1.5. Ergebnisse

Mit der MOBIUS Methodik ist es möglich Personen anhand ihrer Benutzernamen über verschieden soziale Medien hinweg zu identifizieren. Dabei werden Verhaltensmuster modelliert und diese systematisch zu Merkmalen ausgearbeitet. Der minimale Informationsgehalt eines Benutzernamens wird dabei ausgeschöpft um eine größtmögliche Anzahl an weiteren Merkmalen zu erzeugen. Dabei werden die Verhaltensmuster in Muster aufgrund menschlicher Einschränkungen, Exogener Faktoren und Endogener Faktoren kategorisiert, wobei für jeden Faktor in jeder Kategorie eigene Merkmale entwickelt wurden. Die Versuchsergebnisse zeigen Vorteile gegenüber der bisher bekannten Methoden in diesem Bereich.

3.2. Umgehung der Mustererkennung von MOBIUS

Wie unter 3.1 erläutert hat das vorgestellte MOBIUS-Verfahren Schwierigkeiten beim Zuordnen von Benutzernamen, die eine maximale Entropie aufzeigen. Damit sind solche Benutzernamen gemeint, die folgende Eigenschaften aufzeigen:

- die Zeichenlänge der Zeichenkette ist die maximal zulässige
- die Zeichenkette besteht aus einer zufälligen Abfolge aller erlaubten Zeichen
- und sie enthält keine Wiederholungen.

Zudem wird in der Untersuchung erläutert, dass anhand der exogenen Faktoren Rückschlüsse auf die kulturelle Herkunft einer Person geschlossen werden können. Wird aber beispielsweise auf einer Plattform ein gebräuchlicher deutscher Name verwendet und auf einer anderen ein chinesischer oder arabischer Name schlagen die Identifizierungsversuche fehl, da kein Muster zu erkennen ist. Aus diesen Überlegungen lassen sich Testfälle ableiten, die in ihrer Komplexität und ihren Anforderung unterschiedlich sind. Da die generierten Namen mit

zunehmender Komplexität unleserlicher und schlechter zu erinnern sind, werden die vorgestellten Tests in Komplexitätsstufen unterteilt. Das nächste Kapitel zeigt den vollständigen Testplan inkl. negativen Testfällen auf.

3.3. Test-Definitionen

In diesem Kapitel werden die Testfälle beschrieben mit welchen aufgezeigt wird, wie es möglich ist die Mustererkennung von MOBIUS zu umgehen. Dabei werden die Tests in ihrer Komplexität aufsteigend sortiert und anhand ihrer Form kategorisiert. Gleichzeitig bezeichnet die Komplexität, die Wahrscheinlichkeit, nicht wiedererkannt zu werden. Mit anderen Worten: je höher die Komplexität desto höher die Wahrscheinlichkeit, dass MOBIUS kein wiederkehrendes Muster bemerkt.

3.3.1. Kategorie Fehler: Negativ-Tests

Um die Schnittstellendefinition (siehe 5.2.3) zu gewährleisten werden ungültige Anfragen mit einem Fehlercode beantwortet.

ID	Anforderungsszenario	Testszenario	Testdaten	Erwartetes Ergebnis
E-00	Fehlerhafte Anfrage	Fehlerfälle	<leerer POST -oder GET-Request>	HTTP-Statuscode 500
E-01	Fehlerhafte Anfrage	Fehlerfälle	{"UserId": <fehlt> , "Modus":"2"}	HTTP-Statuscode 500
E-02	Fehlerhafte Anfrage	Fehlerfälle	{"UserId":"1", "Modus": <fehlt> }	HTTP-Statuscode 500
E-03	Fehlerhafte Anfrage	Fehlerfälle	{"UserId": "1 "}	HTTP-Statuscode 500
E-04	Fehlerhafte Anfrage	Fehlerfälle	{"Modus": "1 "}	HTTP-Statuscode 500

3.3.2. Kategorie I: Sprechender Name ohne Sonderzeichen

Die einfachste denkbare Namensgenerierung ist ein sprechender Name ohne Trennzeichen und ohne Sonderzeichen und Ziffern.

ID	Anforderungsszenario	Testszenario	Testdaten	Erwartetes Ergebnis
T-00	Erzeuge sprechenden deutschen Namen	Deutscher Name	{"UserId":"1", "Username-Complexi-ty":"10"}	z.B. MarkusMeier

3.3.3. Kategorie II: Sprechender Name inkl. einem Sonderzeichen

Aufbauend auf einem einfachen sprechenden Namen kann der Vor- und Nachname mit einem Sonderzeichen getrennt werden.

ID	Anforderungsszenario	Testszenario	Testdaten	Erwartetes Ergebnis
T-01	Erzeuge sprechenden deutschen Namen mit Sonderzeichen	Deutscher Name	{"UserId":"1", "Username-Complexi-ty":"20"}	z.B. Markus.Meier
T-02	Erzeuge sprechenden deutschen Namen mit Sonderzeichen	Deutscher Name	{"UserId":"1", "Username-Complexi-ty":"30"}	z.B. Markus_Meier

3.3.4. Kategorie III: Sprechender Name inkl. Pre -und/oder Suffix als Zahl

Ausgehend von 3.3.3 werden die Namen zusätzlich mit einem Pre -oder Suffix oder beidem bestehend aus Zahlen erweitert. Außerdem kann das Trennzeichen zusätzlich um eine Zahl erweitert werden.

ID	Anforderungsszenario	Testszenario	Testdaten	Erwartetes Ergebnis
T-03	Erzeuge sprechenden deutschen Namen mit Prefix als Zahl	Deutscher Name	{"UserId":"1", "Username-Complexi-ty":"40"}	z.B. 42Markus.Meier
T-04	Erzeuge sprechenden deutschen Namen mit Suffix als Zahl	Deutscher Name	{"UserId":"1", "Username-Complexi-ty":"50"}	z.B. Markus_Meier42
T-05	Erzeuge sprechenden deutschen Namen mit Pre -und Suffix als Zahl	Deutscher Name	{"UserId":"1", "Username-Complexi-ty":"60"}	z.B. 18Markus_Meier42
T-06	Erzeuge sprechenden deutschen Namen mit Pre -und Suffix als Zahl	Deutscher Name	{"UserId":"1", "Username-Complexi-ty":"70"}	z.B. 18Markus32_Meier42
T-07	Erzeuge sprechenden deutschen Namen mit Pre -und Suffix als Zahl	Deutscher Name	{"UserId":"1", "Username-Complexi-ty":"80"}	z.B. 18Max32_90Meier42

3.3.5. Kategorie IV: Sprechender Name inkl. Pre -und/oder Suffix als Sonderzeichen

Ähnlich wie 3.3.4, mit dem Unterschied dass statt Zahlen Sonderzeichen als Pre -oder Suffix verwendet werden.

ID	Anforderungsszenario	Testszenario	Testdaten	Erwartetes Ergebnis
T-08	Erzeuge sprechenden deutschen Namen mit Pre -und Suffix als Sonderzeichen	Deutscher Name	{"UserId":"1", "Username-Complexity":"90"}	z.B. !?Markus.Meier
T-09	Erzeuge sprechenden deutschen Namen mit Pre -oder Suffix als Sonderzeichen	Deutscher Name	{"UserId":"1", "Username-Complexity":"100"}	z.B. Markus_Meier?!
T-10	Erzeuge sprechenden deutschen Namen mit Pre -oder Suffix als Sonderzeichen	Deutscher Name	{"UserId":"1", "Username-Complexity":"110"}	z.B. !?Markus_Meier=&
T-11	Erzeuge sprechenden deutschen Namen mit Pre -und Suffix als Sonderzeichen	Deutscher Name	{"UserId":"1", "Username-Complexity":"120"}	z.B. !?Max%§_Meier=&
T-12	Erzeuge sprechenden deutschen Namen mit Pre -und Suffix als Sonderzeichen	Deutscher Name	{"UserId":"1", "Username-Complexity":"130"}	z.B. ?!Max%§-_(=90Rose=&

3.3.6. Kategorie V: Namen aus Sonderzeichen

Namen, die ausschließlich aus Sonderzeichen bestehen, stellen mit der höchsten Komplexität die höchste Wahrscheinlichkeit dar von der MOBIUS-Mustererkennung nicht (wieder-)erkannt zu werden. Da für die jeweiligen Webseiten unterschiedliche Anforderungen an einen zulässigen Benutzernamen bestehen muss zusätzlich die jeweilige Konfiguration für diese Webseite bekannt sein. Aus diesem Grund wird ein dritter Parameter benötigt, der aussagt um welche Socia Media Plattform es sich handelt.

ID	**Anforderungsszenario**	**Testszenario**	**Testdaten**	**Erwartetes Ergebnis**
T-13	Erzeuge zufällige Zeichenkette mit Großbuchstaben und max. Länge	Zufällige Zeichenkette	{"UserId":"1", "UsernameComplexity":"140" , "SiteId":"1"}	z.B. AIKEMLOWQP
T-14	Erzeuge zufällige Zeichenkette mit Kleinbuchstaben und max. Länge	Zufällige Zeichenkette	{"UserId":"1", "UsernameComplexity":"150" , "SiteId":"1"}	z.B. aikemlowqp
T-15	Erzeuge zufällige Zeichenkette mit Groß - und Kleinbuchstaben und max. Länge	Zufällige Zeichenkette	{"UserId":"1", "UsernameComplexity":"160" , "SiteId":"1"}	z.B. aIkEmloWQp
T-16	Erzeuge Benutzernamen mit max. Entropie	Zufällige Zeichenkette	{"UserId":"1", "UsernameComplexity":"170" , "SiteId":"1"}	z.B. U8(0§-_.l!?=;.oP

4. Client-Komponente

Im folgenden Kapitel wird zuerst die Analyse der Komponente beschrieben. Aus dieser ergeben sich die Anforderungen, welche im Entwurf beachtet werden müssen. Im letzten Teilabschnitt wird die Implementierung der Client-Komponente beleuchtet.

4.1. Analyse

Die Client-Komponente dient dazu aufzuzeigen, wie eine Benutzerverschleierung über verschiedene Internetseiten hinweg, trotz der vorgestellten MOBIUS-Mustererkennung für Benutzernamen, aussehen könnte. Hierbei ist die Idee, dass der Benutzer eine Erweiterung in seinen Browser integriert, die automatisch reagiert, wenn eine der Erweiterung bekannten Regestrierungsseite aufgerufen wird. Ist dies der Fall, blendet die Komponente ein Dialog-Fenster ein. Der Dialog ermittelt, ob der Benutzer einen generierten Namen in das Formularfeld für Benutzernamen einfügen möchte. Ist dies der Fall, sendet der Client eine Anfrage an den Server ab und erhält einen generierten Benutzernamen. Der Benutzer kann die Komplexitätsstufen der Benutzernamen im Options-Bereich der Erweiterung konfigurieren und aktuelle Konfigurationen für Registrierungsseiten vom Server abrufen. Außerdem kann er wählen, ob jedes Mal ein neuer Benutzername generiert werden soll, oder ob der bereits generierte Namen eingesetzt werden soll.

Aus diesen Voraussetzungen und Überlegungen ergeben sich folgende Anforderungen für die Client-Komponente, welche im Rahmen dieser Arbeit entwickelt werden soll.

4.1.1. Funktionale Anforderungen

Eine Anforderung bezeichnet man als funktional, wenn das ihr zu Grunde liegende Bedürfnis funktional ist, d.h. sich auf Gegenstände der Informationsverarbeitung (Daten, Operationen, Verhalten) bezieht. Für die zu entwickelnde Client-Komponente ergeben sich folgenden funktionalen Anforderungen:

- CF-R-01: Benutzer können im Optionsbereich die Komplexität der generierten Benutzernamen mit einem Schieberegler konfigurieren.

- CF-R-02: Die Komponente erkennt automatisch Registrierungsseiten. Wird eine solche Seite erkannt, erscheint ein Dialogfenster.
- CF-R-03: Bei Bestätigung des Dialogfensters wird ein Benutzername generiert und in das Formularfeld für den Benutzernamen eingesetzt. Andernfalls wird keine weitere Funktion ausgeführt.
- CF-R-04: Ein Benutzer kann konfigurieren, ob ein Benutzername jedes mal neu generiert wird oder der vorhandene geladen werden soll.
- CF-R-05: Ein Benutzer kann die Konfiguration der Registrierungsseiten manuell aktualisieren.

Neben den funktionalen Anforderungen werden auch nichtfunktionale Anforderungen wie folgt definiert.

4.1.2. Nichtfunktionale Anforderungen

Eine Anforderung wird als nicht-funktional bezeichnet, wenn das ihr zu Grunde liegende Bedürfnis eine nicht gegenständliche Eigenschaft ist. Für die Client-Komponente werden folgenden nichtfunktionalen Anforderungen definiert:

- CN-FR-01: Der GUI-Bereich der Komponente muss in HTML5 realisiert sein.
- CN-FR-02: Dynamische Funktionen müssen in JavaScript implementiert werden.
- CN-FR-03: Die Möglichkeit des Test-Driven Developments muss gegeben sein.
- CN-FR-04: Die Manipulation von HTML5-Elementen, insbesondere Formularfelder, soll möglichst komfortabel gestaltet sein.
- CN-FR-05: Die Kommunikation mit dem Server findet über AJAX-Anfragen statt.
- CN-FR-06: Die Client-Konfiguration muss auf dem Client gespeichert werden.
- CN-FR-07: Die Registrierungsseiten-Konfiguration muss mittels Serveraufruf aktualisierbar sein.
- CN-FR-08: Soll der vorhandene Benutzernamen geladen werden, wird dieser vom Server abgerufen.

- CN-FR-09: Die eingesetzten Komponenten und Technologien, die zur Erstellung benötigt werden, sollen eine möglichst hohe Verbreitung haben. Damit werden Einarbeitungszeiten für eine Weiterentwicklung durch eine Community oder andere Entwickler verkürzt.
- CN-FR-10: Sowohl die eingesetzten externen Komponenten als auch die entwickelte Komponente selbst müssen gut dokumentiert sein.
- CN-FR-11: Um ein Mindestmaß an Sicherheit zu gewährleisten, muss die Kommunikation zwischen dem Client -und Serverkomponente geschützt sein. Somit sollen Angriffe von außerhalb erschwert werden.
- CN-FR-12: Die einzelnen externen Module müssen einfach zu aktualisieren sein.
- CN-FR-13: Bei der Entwicklung der Client-Komponenten muss auf Modularisierung geachtet werden, so dass einzelne Module und/oder Komponenten einfach erweitert oder ausgetauscht werden können.

Während die funktionalen Anforderungen vom Benutzer als solche wahrgenommen werden, sind nichtfunktionale Anforderungen Implementierungsdetails, die dem Benutzer zum überwiegenden Teil vorenthalten bleiben.

4.2. Entwurf

Die aus der Analyse gewonnen Informationen werden im Entwurf aufbereitet. Anhand der Anforderungen werden Entscheidungen im Bezug auf den späteren Aufbau der Client-Komponente getroffen. Erkenntnisse und Erfahrungen aus früheren Projekten spielen bei den einzelnen Entscheidungen ebenfalls eine Rolle. Oft werden hierbei falsche Annahmen getroffen, welche im späteren Verlauf der Implementierung zu Schwierigkeiten führen. Im Rahmen dieser Arbeit wurde deshalb das Verfahren des Test-Driven Developments gewählt, um so möglichst kleine iterative Schritte zu ermöglichen, Probleme frühstmöglich zu identifzieren und zu beseitigen. Aus diesem Grunde stehen Tests und die damit verbundene Infrastruktur im Fokus des Entwurfs.

4.2.1. Tests

Aus der Analyse ergeben sich Testfälle bezüglich der Funktionalität, welche nachfolgend als Regressiontests formuliert werden. Einige dieser Testfälle können ebenfalls als Integrationstest betrachtet werden. Durch diese Tests soll sichergestellt werden, dass das Mindestmaß an Funktionalität auch nach Änderungen an der Client-Komponente weiterhin erhalten bleibt. Die Tests sollen nicht automatisch ausgeführt werden, da sie die lokale Client-Konfiguration für einzelne Testbestandteile zurücksetzen und somit die aktuelle Konfiguration überschreiben. Auf Mocks wurde absichtlich verzichtet, da diese die Tests unnötig kompliziert gemacht hätten und keinen Mehrwert für diese Arbeit geboten hätten. Daher werden die benötigten Test-Daten statisch für die jeweiligen Tests definiert.

Modul Client-Komponente Testdefinitionen

ID	Anforderungsszenario	Testszenario	Testdaten	Erwartetes Ergebnis
TC-1	Stelle sicher, dass Klassen-Variablen zurückgesetzt werden	Klassen-Variablen zurücksetzen	n/a	Alle Klassen-Variablen sind zurückgesetzt
TC-2	Stelle sicher, dass ein neuer Benutzer eine neue UserID erhält	UserID generieren	n/a	UserID als Klassen-Variable verfügbar
TC-3	Erzeuge zufälligen Benutzernamen	Zufällige Zeichenkette	{"UserId":"1", "Username-Complexity":"30" , "SiteId":"1"}	z.B. {"name ":"Markus_Meier "}
TC-4	Setzten der Klassen-Variable URL funktioniert	URL setzen	setCurrentUrl (www.testUrl.org)	getCurrentUrl() == www.testUrl.org

Modul Konfiguration Testdefinitionen

ID	Anforderungsszenario	Testszenario	Testdaten	Erwartetes Ergebnis
TC-5	Stelle sicher, dass Konfiguration zurückgesetzt wird	Konfiguration wird zurückgsetzt	n/a	Alle Klassen-Variablen der Konfiguration sind zurückgesetzt
TC-6	Stelle sicher, dass Konfiguration gesetzt wird	Konfiguration setzen	n/a	Alle Klassen-Variablen der Konfiguration sind gesetzt
TC-7	Stelle sicher, dass Benutzernamen-Komplexität aktualisiert wird	Benutzernamen-Komplexität Aktualisierung	setUsername Complexity(30)	getUsernameComplexity == 30
TC-8	Stelle sicher, dass Aktualisieren des Lade BenutzernamenFlags funktioniert	LadeBenutzernamen Flag ändern	setLoadExisting Name (true)	getLoadExisting Name == true

4.2.2. Designentscheidungen

Aus den angeführten Tests und genannten Anforderungen ergeben sich eine Reihe von Entscheidungen, die getroffen werden müssen und den Entwurf weiter beeinflussen. Die folgenden Abschnitte zeigen auf, welche Entscheidungen zum Entwurf beigetragen haben.

Persistenter Speicher

Aus den Anforderungen ergibt sich, dass die Möglichkeit einer persistenten Speicherung von Daten gewährleistet werden muss. Prinzipiell bieten sich bei der Entwicklung einer Google Chrome Extension hierbei zwei Möglichkeiten an:

- chrome.storage [12]
- HTML5 Web Storage [30]

Der chrome.storage ist eine API, welche von Google Chrome angeboten wird. Mit ihr ist es möglich Daten direkt zu synchronisieren und TAB-übergreifend Daten zu persistieren. Das

Speichern findet asynchron statt, so dass Skripte nicht blockiert werden. Deshalb bietet das asynchrone Speichern einen Performance-Gewinn gegenüber dem synchronen. Bei der Implementierung muss das asynchrone Verhalten beachtet werden. Der HTML5 Web Storage oder LocalStorage ist auf max. 5 MB Speicher begrenzt und ermöglicht kein TAB-übergreifendes Speichern, da die Daten nur im aktuellen Kontext verfügbar sind. Das TAB-übergreifende Speichern und Auslesen ist zwingend für die Entwicklung der Client-Komponente erforderlich. Aufgrund der begrenzten Möglichkeiten vom HTML5 Storage im Bezug auf die Entwicklung einer Chrome Extension fällt die Entscheidung auf die Verwendung des chrome.storages.

Sicherheit

Um ein mindestmaß an Sicherheit bzgl. des Datenaustauschs zwischen Server -und Client-Komponente zu gewährleisten ist es notwendig, dass der Server nur authorisierte Anfragen beantwortet. Aufgrund der einfachen Implementierung wurde sich für die Basic Access Authentication [4] entschieden.

Kommunikation mit Server

Die Kommunikation mit dem Server, auf dem aus Performance-Gründen und dem Halten einer Datenbank die Namen generiert werden, muss sichergestellt werden. Die Aufrufe des Clients an den Server finden dabei als AJAX-Request statt. Dabei kann zwischen den Datenformaten XML, Plain-Text oder JSON gewählt werden. Aufgrund der einfachen Struktur von JSON-Objekten und der schnell anwachsenden Größe von XML-Objekten wird sich für JSON entschieden. Eine Plain-Implementierung würde die Kommunikation aufgrund des notwendingen Parsings komplexer machen.

Bereitstellung des Testframeworks

Die jeweiligen Unit-Tests müssen im Kontext einer Chrome Extension funktionieren, damit das Testen des chrome.storage gewährleistet werden kann. Außerdem ist so ein besseres Vertesten der einzelnen Bestandteile der Client-Komponente möglich. Auf den Einsatz eines Mocking-Frameworks zur Datensimulation wurde aufgrund der steigenden Komplexität verzichtet. Stattdessen wird ein Konfigurations-Objekt verwendet, welches für die Tests verwendet wird. Es wird QUnit als Testframework eingesetzt werden, da es eine modulgetriebe Entwicklung ermöglicht und weit verbreitet ist.

4.2.3. Komponentendiagramm

Das folgende Komponentendiagramm dient zur Visualisierung der einzelnen Bestandteile der Client -und Server-Komponente.

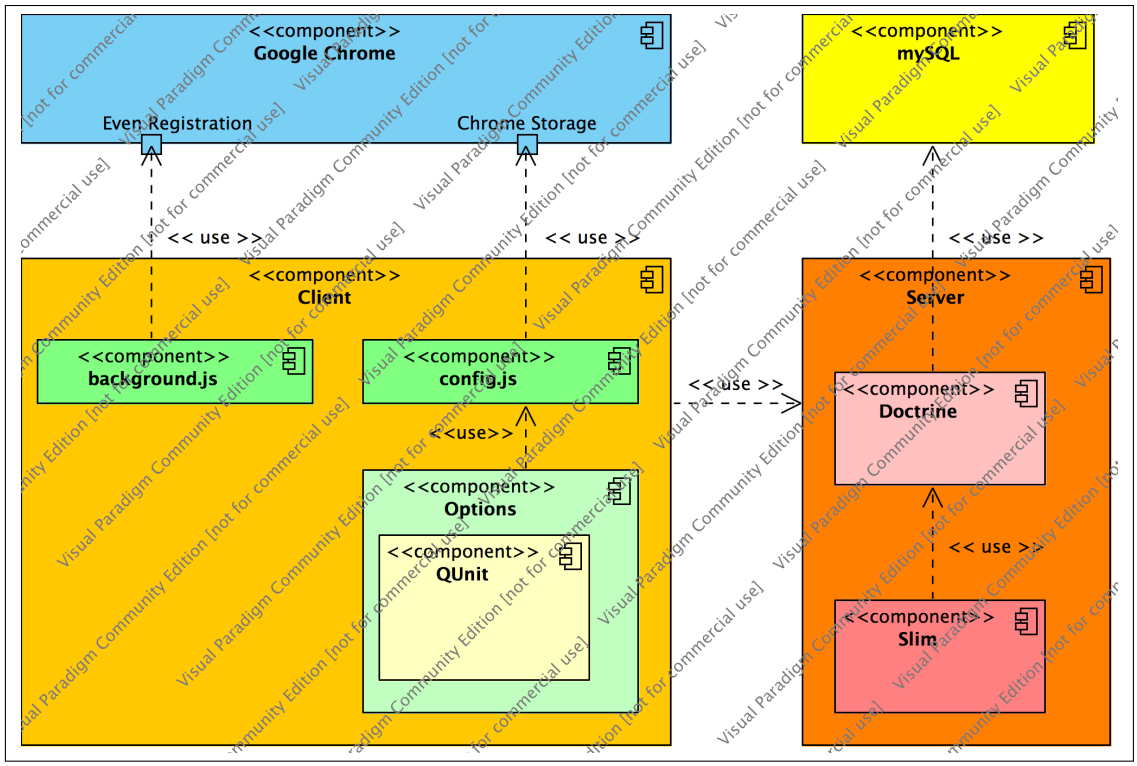

Abbildung 4.1.: Gesamtansicht der Komponenten

Die von Google Chrome angebotene Event Registration dient dazu die einzelnen Funktionen, die bei Events ausgelöst werden sollen, zu registrieren. Diese Funktionen werden in der Datei background.js implementiert. Die API des chrome.storage wird für die Persistierung des Konfigurations-Objektes benötigt, welches die Datei config.js implementiert und stark an ein POJO (Plain-Old-Java-Object) angelehnt ist, da es im Wesentlichen nur aus Getter -und Setter-Methoden besteht. Um die Unit-Tests im selben Kontext wie die Client-Komponente auszuführen, wurden diese in den Konfigurationsbereich der Komponente (Options) in Form von QUnit-Tests integriert.

Die Kommunikation mit dem Server findet über JSON-Aufrufe statt. Hierbei werden im wesentlichen drei Schnittstellen benötigt:

- Server Konfiguration abrufen: wird benötigt um die aktuellen Konfigurationen der Registrierungsseiten vom Server abzurufen
- Benutzernamen generieren: erstellt anhand der übermittelten Anfragedaten des Clients einen Benutzernamen auf dem Server und übermittelt den generierten Namen an den Client
- UserID: erstellt einen HASH-Wert als eindeutige Benutzerkennung, sofern diese noch nicht auf dem Client vorhanden ist

Auf die Schnittstellen des Servers wird unter 5.2.3 detailliert eingegangen.

Die Server-Komponente besteht im Wesentlichen aus zwei Subkomponenten. Das Slim-Framework dient zur Bereitstellung der REST-Schnittstellen, die vom Client zum Datenaustausch genutzt werden. Das Routing, also die Zuordnung von URL-Aufrufen zu den jeweiligen ausgeführten Funktionen, wird hier ebenfalls definiert. Der Doctrine OR-Mapper dient als Abstraktionsschicht für die Datenschicht. Durch die Verwendung eines OR-Mappers wird die Implementierung innerhalb des Servers weitestgehend von der zugrundeliegenden Datenbank entkoppelt, was den Austausch der Datenbank vereinfacht. Außerdem ist so objektorientiertes Arbeiten mit Datenbank-Entities innerhalb der Server-Komponente möglich, ohne Wissen über die genaue Struktur der Datenbank zu besitzen. Auch im Bereich der Skalierung macht das Nutzen von OR-Mappern durchaus Sinn, da die Integration von Load Balancing Mechanismen vereinfacht wird.

4.3. Implementierung

Nach der Erstellung des Entwurfs kann mit der Implementierung der Client-Komponente begonnen werden. Die dazu verwendeten Technologien sind unter 2.2.2 beschrieben.

4.3.1. Installation

Bei der Installation einer Chrome Extension wird eine sogenannte Manifest-Datei eingelesen, die gewissermaßen einen Vertrag zwischen Erweiterung und dem Browser selbst darstellt. Dabei werden neben trivialen Daten, wie beispielsweise dem Namen der Erweiterung, auch die Rechte definiert, die zur Laufzeit benötigt werden (*permissions*). Außerdem werden mit Hilfe des sogenannte background JavaScripts (*background*), die Funktionen registriert, welche nach eingetretenen Ereignissen ausgelöst werden sollen. Zudem ist es möglich eine Options-Seite bzw. Konfiguration-Seite zu definieren (*options_page*).

```
[...]
  "name": "Anonymous name generator",
  "background": {"scripts": ["javascripts/background.js"]},
  "options_page": "html/options.html",
  "permissions": [
    "http://thesis.mymoshpit.net/",
    "https://thesis.mymoshpit.net/",
    "storage",
    "tabs",
    "http://*/*", "https://*/*",
    "activeTab"
  ]
[...]
```

Listing 4.1: Ausschnitt manifest.json

4.3.2. Umsetzung

Während sich die Umsetzung selbst im Wesentlichen auf das Schreiben von Tests und anschließender Implementierung konzentrierte, ergaben sich im Zuge der Integration des Quellcodes in eine Google Chrome Extension einige Fragestellungen, die es zu lösen galt. Dabei stand insbesondere die Frage im Fokus, wie sich mehrere JavaScript-Bibliotheken in den Kontext eines Browser-Tabs und somit in den Kontext der aktuellen Seite einladen lassen um anschließend die Formularfelder auszulesen oder mit Daten zu füllen. Die Lösung besteht daraus, nacheinander geschachtelte API-Aufrufe der Chrome-Engine auszuführen.

```
[...]
chrome.tabs.executeScript(null,
  {file: "javascripts/jquery-2.1.0.min.js"},
  function() {
    chrome.tabs.executeScript(null,
    {file: "javascripts/jquery-ui-1.10.4"},
    function() {
      [...]
  });
});
[...]
```

Listing 4.2: Ausschnitt background.js

Anstatt die unterschiedlichen JavaScript-Bibliotheken nacheinander einzufügen, ist es auch vorstellbar, alle Funktionen in einer Datei zu kombinieren. Aufgrund der Intransparenz wurde auf diesen Weg im Rahmen dieser Arbeit verzichtet.
Das Testen von AJAX-Funktionen stand ebenfalls im Fokus dieser Arbeit. Durch die asynchronen Aufrufe lassen sich Abläufe schwieriger Testen und bedurften einer gesonderten Behandlung. Um die Testfälle möglichst konsistent und lesbar zu halten wurde sich deshalb dazu entschieden bei den jeweiligen AJAX-Aufrufen, mittels Chaining auf den AJAX-JQuery Objekte zu arbeiten. Dies hat den großen Vorteil, dass direkt auf den Objekten die Asserts durchgeführt werden können und mögliche Fehler direkt ausgegeben werden können. Parallel dazu wurde z.T. mit Timeout -respektive Sleep -oder Warte-Funktionen gearbeitet. Das Chaining, mit welchem die Done-Funktion der AJAX-Aufrufe aufgerufen werden kann, vereinfacht das Schreiben und Lesen der Testfälle ebenfalls.
Anhand der Konfigurationen der Registrierungsseiten, die innerhalb der Client-Komponente gespeichert werden, werden JQuery-Selektoren verwendet um die generierten Benutzernamen in die Registrierungsformulare einzusetzen.

Options-Page

Bei der Options-Page handelt sich um eine gewöhnliche HTML5-Seite, die zusätzlich um einige JavaScript Funktionen erweitert wurde. Damit nicht bei jedem Aufruf der Options-Page die QUnit-Tests ausgeführt werden, wurde ein Button implementiert, der mittels AJAX-Aufruf den Aufruf der QUnit-Testseite (qunit.html) in einen iFrame nachlädt. So wird sichergestellt, dass die Konfiguration der Client-Komponente nicht versehentlich überschrieben wird.

4.3.3. Verifikation

Mittels Blanket.js [14] lässt sich QUnit derart erweitern, dass sich die Testabdeckung respektive Code Coverage ermitteln lässt. Die benötigten Testdaten selbst wurden statisch in ein Konfigurations-Objekt ausgelagert. Während für das Konfigurations-Objekt (config.js) eine Testabdeckung von 100% erreicht wurde, wurde für die Client-Komponente (vusa.js) selbst eine Testabdeckung von 82,61% ermittelt. Insgesamt ergibt sich so eine Gesamtabdeckung von 92,59%. Der geringere Umfang bei letzterer ist damit zu begründen, dass es nicht ohne weiteres möglich ist nach dem Auftreten des JavaScripts Dialogs den ausgeführten Code automatisiert zu testen. Zudem müsste eine Möglichkeit geschaffen werden die Registrierungsseite(n) inkl. Formular zu simulieren damit die Selektoren getestet werden können. Mittels Selenium wäre es möglich ein derartiges Szenario aufzubauen. Im Rahmen dieser Arbeit wurde dies aufgrund des Aufwands aber nicht berücksichtigt.

Blanket.js results	Covered/Total Smts.	Coverage (%)
1. chrome-extension://dfhlkdbiopkofennanljcgodgabkiold/javascripts/config.js	31/31	100 %
2. chrome-extension://dfhlkdbiopkofennanljcgodgabkiold/javascripts/vusa.js	19/23	82.61 %
Global total	50/54	92.59 %

Abbildung 4.2.: Code Coverage der Client-Komponente

5. Server-Komponente

Dieses Kapitel beschreibt im ersten Abschnitt die Analyse der Server-Komponente. Anschließend werden die resultierenden Anforderungen in Form des Entwurfs ausgeführt. Das Ende des Kapitels behandelt die Implementierung und die Tests der Komponente.

5.1. Analyse

Üblicherweise werden im Zuge der Analyse zusammen mit dem zu befragenden Kunden dessen Anforderungen an das Software-Produkt erörtert. Dem Umstand folge tragend, dass im Rahmen dieser Arbeit keine Kundenbefragung statt finden kann, da als Grundlage nur die wissenschaftliche Arbeit über MOBIUS vorhanden ist, muss ein alternativer Weg im Rahmen der Analyse verfolgt werden um die Anforderungen zu definieren. Nach Gesprächen mit Kommilitonen und IT-Sicherheits-Expertern auf Konferenzen und anderen Veranstaltungen, stellt sich zunehmend heraus, dass die Analyse zunehmend den Charakter einer Spezifikation hat. Dies ist sowohl auf Serverseite, wie auch für die Analyse der Client-Komponente unter 4.1 festzustelllen. Somit ist das Paper über die Entwicklung von MOBIUS Grundlage der Analyse. Kombiniert mit den Überlegungen aus den geführten Diskussionen entwickelt sich eine Menge von Anforderungen und die Idee, die Mustererkennung zu umgehen. Besonder Fokus liegt hierbei im Weiteren auf der Wiederverwendbarkeit der zu entwickelnden Komponenten, und darauf, dass weitere Entwickler möglichst geringe Einarbeitungszeiten haben. So sollen Einstiegshürden überwunden werden und ein größere Zielgruppe von weiteren potentiellen Entwicklern angesprochen werden.

Nachfolgend werden die funktionalen und nichtfunktionalen Anforderungen an die Server-Komponente definiert.

5.1.1. Funktionale Anforderungen

Nachfolgend werden die funktionalen Anforderungen für die Server-Komponente definiert.

- S-FR-01: Unabhängig vom aufrufenden Client kann der Server anhand der URL ermitteln, welche Funktion aufgerufen werden soll.

- S-FR-02: Der Server kann Namen unterschiedlicher Komplexität generieren.
- S-FR-03: Der Server kann eindeutige UserIDs erzeugen.
- S-FR-04: Der Server bietet eine Schnittstelle an, welche die Konfigurationen der einzelnen Registrierungsseiten zurückgibt.

Die Anforderungen sind so gewählt das sie unabhängig von dem kommunizierenden Client stehen können.

5.1.2. Nichtfunktionale Anforderungen

Aus Abschnitt 2.3.1 ergeben sich die folgenden nichtfunktionalen Anforderungen:

- S-NFR-01: Die Server-Komponenten soll als REST-Service realisiert werden, so dass über den Aufruf der URLs die jeweilige Funktion des Servers angesteuert werden kann.
- S-NFR-02: Die Skalierbarkeit soll mit Hinblick auf einen möglichen Load-Balancer sichergestellt sein.
- S-NFR-03: Mit Hilfe einer Datenbank sollen sprechenden Namen generiert werden können.
- S-NFR-04: Die zu verwendende Infrastruktur soll eine möglichst hohe Verfügbarkeit und Verbreitung haben.
- S-NFR-05: Die eingesetzten Installationsroutinen sollen frei verfügbar und gut dokumentiert sein.
- S-NFR-06: Es soll möglich sein, die einzelnen Module der Server-Komponente testgetrieben zu entwickeln.
- S-NFR-07: Die für den Server eingesetzten Komponenten sollen als Open Source vorliegen.

sowie

- S-NFR-08: Der Server gibt bei den jeweiligen Funktionsaufrufen Datenobjekte wieder, die möglichst datensparsam dargestellt werden sollen.
- S-NFR-09: Um eine Abstraktion von der Datenschicht zu erreichen, soll ein OR-Mapper eingesetzt werden.

- S-NFR-10: Der Server kann anhand der Anfrage die Instanz der Chrome Erweiterung erkennen, sofern die notwendigen Daten übermittelt wurden.
- S-NFR-11: Für den Server ist es unerheblich, ob der Aufruf als AJAX-Aufruf oder manuell über den Browser geschieht. Der Funktionsumfang soll unabhängig vom Client gewährleistet sein.

Die letzte Anforderung S-NFR-11 betont hierbei die Unabhängigkeit vom Server gegenüber dem Client.

5.2. Entwurf

Grundlegende Überlegung bei dem vorliegenden Entwurf ist es, dass der zu betreibende Implementierungsaufwand auf ein Minimum reduziert werden soll und bereits vorhandene Lösungen miteinander integriert werden sollen. Gerade letzteres ist bei der großen Anzahl der existierenden Lösungen eine große Herausforderung. Der Fokus dieser Arbeit liegt also auf dem Integrativen und nicht Implementierungstechnischen.

5.2.1. Tests

Um die Funktionalität der Server-Komponente bei jedem Iterationsschritt zu gewährleisten soll diese, wie die Client-Komponente, testgetrieben entwickelt werden. Deshalb stehen auch hier die Testdefinitionen im Fokus des Entwurfs. Die Testfälle testT00-testT16 sind aufgrund der Redundanz nicht vollständig aufgeführt. Es ändert sich nur die jeweilige usernameComplexity. Aus den Test-Definitionen im Konzept 3.3 werden folgenden Testfälle ermittelt:

Modul Server-Komponente Testdefinitionen

ID	Anforderungsszenario	Testszenario	Testdaten	Erwartetes Ergebnis
testE00	Fehlende POST-Parameter in Request	Fehlerszenario	{}	HTTP-Status Code 500
testE01	Fehlende UUID in Request	Fehlerszenario	{"platformname": "https://www.stumbleupon.com/signup", "usernameComplexity":"10"}	HTTP-Status Code 500
testE02	Fehlende usernameComplexity in Request	Fehlerszenario	{"platformname": "https://www.stumbleupon.com/signup", "uuid":"52cdb60ced241"}	HTTP-Status Code 500
testE03	Fehlende usernameComplexity und Platform in Request	Fehlerszenario	{"uuid": "52cdb60ced241"}	HTTP-Status Code 500
testE04	Fehlende UUID und Platform in Request	Fehlerszenario	{"username Complexity": "10"}	HTTP-Status Code 500
testT00 - testT16	Benutzernamen erzeugen	Erzeuge Benutzernamen	{"platformname": "https://www.stumbleupon.com/signup", "uuid":"52cdb60ced241", "username Complexity": "10 - 170"}	Benutzername, siehe 3.3 für erwartetes Ergebnis

5.2.2. Designentscheidungen

Die Überlegungen für den Einsatz einer LAMP-Infrastruktur sind bereits unter 2.3.2 behandelt worden. Im Folgenden wird auf das Entwurfsmuster sowie die Überlegungen eingegangen, welche zur Entscheidung der eingesetzten Frameworks beigetragen haben.

Model View Controller

Um einen flexiblen Entwurf zu erreichen, bei welchem im weiteren Verlauf Änderungen und Erweiterungen keine größeren Aufwände verursachen, bietet sich ein Schichtenmodell an, bei dem Zuständigkeiten voneinander entkoppelt sind und keine ungewollten Abhängigkeiten auftreten. Außerdem soll so der Wert der Wiederverwendbarkeit gesteigert werden. Da sich bei steigender Anzahl der Schichten der Detailierungsgrad zunehmend verliert und somit die Wiederverwendbarkeit erschwert wird, wurde sich dazu entschieden ein dreischichtiges Model zu wählen. In Form des Model View Controller (MVC) Ansatzes findet ein geeigneter Grad an Entkoppelung statt. Dabei wird die Datenschicht in Form von sogenannten Models oder Entities repräsentiert, so dass kein Wissen über die Datenschicht bekannt sein muss und diese bei Bedarf ausgetauscht werden kann ohne die eigentliche Implementierung zu ändern. Die Controller spiegeln die eigentliche Anwendungslogik wieder, während die Views die Präsentationsschicht darstellen. Da es sich bei der Server-Komponente um eine Komponente handelt, die im wesentlichen Daten im JSON-Format ausgibt, ist letzteres trivial. Dennoch sollte dies bei der Implementierung beachtet und Logik und Präsentation nicht vermischt werden.

Slimframework

Im PHP-Umfeld existieren eine Reihe von PHP-REST-APIs. Eine gute Übersicht bietet [25]. Ausschlaggebend bei dem Einsatz des Slimframeworks sind vor allem seine Leichtgewichtigkeit und die guten Konfigurationsmöglichkeiten. Des Weiteren wird es mit PHPUnit testgetrieben entwickelt, ist vollständig Open Source und unter der MIT-Lizenz verfügbar. Andere Frameworks bieten einen größeren Funktionsumfang, allerdings benötigen sie auch größere Einarbeitungszeiten und besitzen zum Teil sehr komplexe Konfigurationsmechanismen.

Doctrine

Das Doctrine-Project, insbesondere dessen OR-Mapper, erfreuen sich großer Beliebtheit. So wird dieser in einigen der populärsten PHP-Frameworks eingesetzt. Dazu gehören: Zend Framework [33], CodeIgniter [9] und CakePHP [6]. Dies führt zu einer großen Verbreitung. Der

Fokus liegt insbesondere auf Geschwindigkeit und Reichhaltigkeit der eingesetzten Funktionen. Ein anderes populäres Open Source Projekt in diesem Zusammenhang ist Propel [8]. Allerdings ist dieses Projekt weniger aktiv, die Dokumentation weniger umfangreich und weniger populär.

5.2.3. Schnittstellendefinitionen

Die Schnittstellendefinition dient dazu, die Kommunikation zwischen Server und Client sicherzustellen. Dieser Datenaustausch findet über JSON-Objekte statt. Der Server ermittelt anhand der aufgerufenen URL respektive des aufgerufenen Pfades die gewünschte Funktion. Während die Funktionen, welche zur Namensgenerierung verwendet wird, weitere POST -oder GET-Parameter benötigen, ist es bei anderen nicht nötig weitere Daten des Clients zu erhalten. Nachfolgend werden die drei von der Server-Komponente beschriebenen Schnittstellen beschrieben. Zusammen mit den unter 5.2.1 aufgeführten Tests ergibt sich ein Gesamtbild, wie diese zu nutzen sind.

Namensgenerierung

Die Namensgenerierung wird unter dem Pfad */create/name* angeboten. Da für die verschiedenen Plattformen unterschiedliche Regeln bezüglich der maximalen Länge und Sonderzeichen gelten, muss diese mit angegeben werden. Außerdem muss mandatorisch eine UUID sowie die gewünschte Komplexität des Benutzernamens angegeben werden. Letzterer muss eine Zahl zwischen 10 und 170 sein. Dabei wird in zehner Schritten vorgegangen, kleinere Schritte sind nicht zulässig. Die UUID ist notwendig damit eine Benutzername-Plattform Relation in der Datenbank hinterlegt werden kann. Wird der optionale Parameter *"loadExistingName":"true"* verwendet, so wird der vorhandene Name, sofern vorhanden, zurückgegeben. Andernfalls wird ein neuer generiert und auf dem Server hinterlegt.

```
1 {"platformname":"https://www.stumbleupon.com/signup\",
2 "uuid":"52cdb60ced241",
3 "usernameComplexity":(10-170)}
```

Listing 5.1: Anfrage des Clients zur Namensgenerierung

```
1 { "name":"GENERATED_NAME" }
```

Listing 5.2: Antwort des Servers auf Namensgenerierung

Konfigurationen abrufen

Um die verschiedenen Konfigurationen der Registrierungsseiten auf dem Client zu aktualisieren und neue hinzuzufügen, ist es möglich, die aktuell auf dem Server hinterlegten Konfigurationen abzurufen. Dabei werden keine weiteren Daten von dem Client benötigt. Der Pfad um alle Konfigurationen von dem Server zu laden lautet: */platform/list*.

```
[
  [...]
  {
    "id":5,
    "domain":"stumbleupon.com",
    "minCharacters":2,
    "maxCharacters":15,
    "registrationPage":"https:\/\/www.stumbleupon.com\/signup",
    "jquerySelectorUsername":"#username",
    "jquerySelectorFirstname":"",
    "jquerySelectorLastname":""
  },
  [...]
]
```

Listing 5.3: Antwort des Servers bei Abruf der Konfigurationen

Unique UserID generieren

Für die Client-Komponente ist es notwendig einmalig eine Uniqe UserID zu generieren, damit die Server-Komponente später die Instanz zuordnen kann. Eine UserID wird beim Aufruf des Pfades */uuid/create* erstellt.

```
{"uuid":"53bae0a8e0888"}
```

Listing 5.4: Antwort des Servers bei UUID generieren

5.2.4. Datenschicht

Die Datenbank besteht insgesamt aus fünf Tabellen. Die Tabellen *first_names* und *last_names* enthalten Vor -und Nachnamen, die zur Generierung der sprechenden Namen benötigt werden. Um später die gespeicherten Namen einem Benutzer zuordnen zu können wird für jeden Benutzer eine eindeutige Benutzerkennung erstellt, die in der Tabelle *user* gespeichert wird. Wird ein neuer Benutzername für einen Benutzer generiert, folgt die Speicherung der Relation

zu der entsprechenden Platform in *user_2_platform*. So ist es später möglich, den bereits generierten Namen dem Benutzer, anhand seiner Kennung, einer Registrierungsseite zuzuordnen. Die Tabelle *platform* enthält die Konfigurationen der Registrierungsseiten. Hier werden die URLs und die JQuery-Selektoren für die jeweiligen Formularfelder gepflegt. Außerdem werden die minimale und maximale Länge der zulässigen Benutzernamen hinterlegt, da diese Informationen zur Generierung der Benutzernamen, insbesondere solcher mit maximaler Entropie, notwendig sind.

Das folgende Entity-Relationship-Modell stellt die Datenschicht und deren Tabellen im Detail dar.

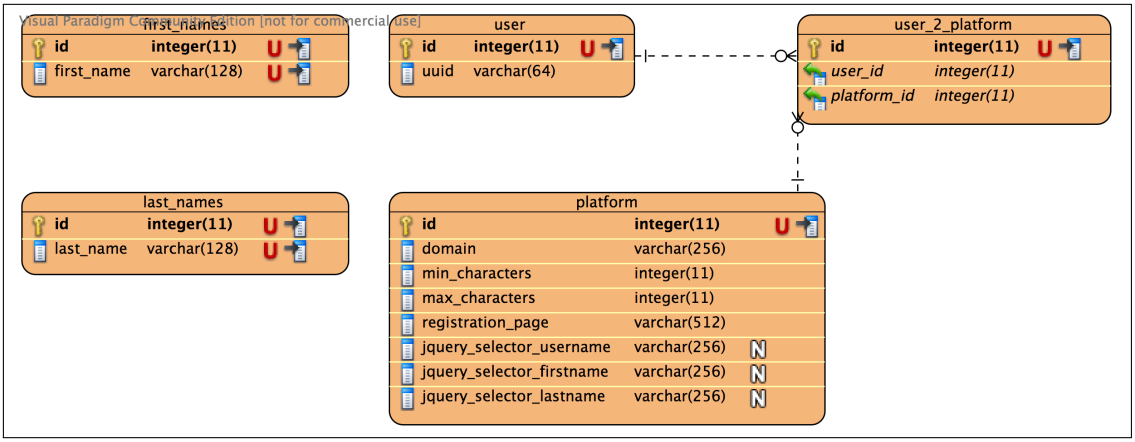

Abbildung 5.1.: Datenschicht der Server-Komponente

5.3. Implementierung und Tests

Im folgenden Kapitel wird auf die Realisierung der Server-Komponente eingegangen. Ähnlich wie bei der Umsetzung der Client-Komponente stehen dabei die Testfälle im Fokus.

5.3.1. Implementierung

Die Server-Komponente integriert das Slimframework, welches dazu dient den REST-Service anzubieten sowie einer Datenabstraktsionsschicht, welche mit dem Doctrine OR-Mapper realisiert wird. Die eigentliche Implementierung konzentriert sich im wesentlichen auf die Integration des Slimframeworks und des OR-Mappers. Da jede der beiden Komponenten einen eigenen ClassLoader verwendet und der Konfigurationsaufwand so gering wie möglich gehalten werden sollte, wird eine Applikations-Klasse implementiert, welche die beiden Komponenten initialisiert und kapselt. Zur Erstelleung der Entities oder Modelle, welche für den OR-Mapper benötigt werden, wird das in Doctrine enthaltene Command Line Interface Tool verwendet. Mit diesem werden zunächst YAML-Dateien [2] erstellt, aus welchen dann im zweiten Schritt die PHP-Dateien generiert werden.

```
$ php doctrine.php orm:convert-mapping
      --namespace="Entities\\"
      --from-database
      yml ../app/Mapping/yml
$ php doctrine.php orm:generate-entities ../app
```

Listing 5.5: CLI Kommandos

Controller

Ein weiterer Schwerpunkt ist die Umsetzung der benötigten Controller. Jeder Controller wird von einem Interface abgeleitet, so dass die Authentifizierung nur einmalig implementiert werden muss. Im Folgenden wird auf die einzelnen Controller und ihre Aufgaben eingegangen.

- ImportController: der ImportController wird dazu verwendet Namenslisten von Wikipedia zu parsen und anschließend zu importieren. Diese Namen werden für die Generierung der sprechenden Namen benötigt. Insgesamt wurden so 4.731 Vornamen und 912 Nachnamen importiert.
- NameController: zuständig für die Generierung der Namen ist der NameController. Er enthält alle Implementierungen für die jeweiligen Komplexitätsstufen.

- PlatformController: dieser Controller wird verwendet um die aktuellen Konfigurationen der Registrierungsseiten abzurufen.
- UuidController: mit diesem Controller werden die UniqueUserIDs erzeugt, die bei der ersten Namensgenerierung generiert und dem jeweiligen Benutzer zugeordnet werden.

Alle Controller überschreiben jeweils die Methode *_initRoutes*, welche bei der Initialisierung aufgerufen wird und die Routen respektive Pfade registriert.

5.3.2. Tests

Für das Durchführen der testgetriebenen Entwicklung wird PHPUnit verwendet. Um abschließend eine Aussage über die Testabdeckung treffen zu können wird neben PHPUnit auch Xdebug verwendet, da dies Voraussetzungen für das PHP_CodeCoverage Framework sind. So ist es möglich ein Reporting durchzuführen. Bezüglich der Controller lässt sich sagen, dass eine Gesamt-Testabdeckung von 86.10% erreicht wurde, bei den übrigen Klassen lässt sich ebenfalls eine hohe Testabdeckung (im Schnitt mit mehr als 70%) verzeichnen. Die vollständige Übersicht des Reports findet sich im Anhang unter A.

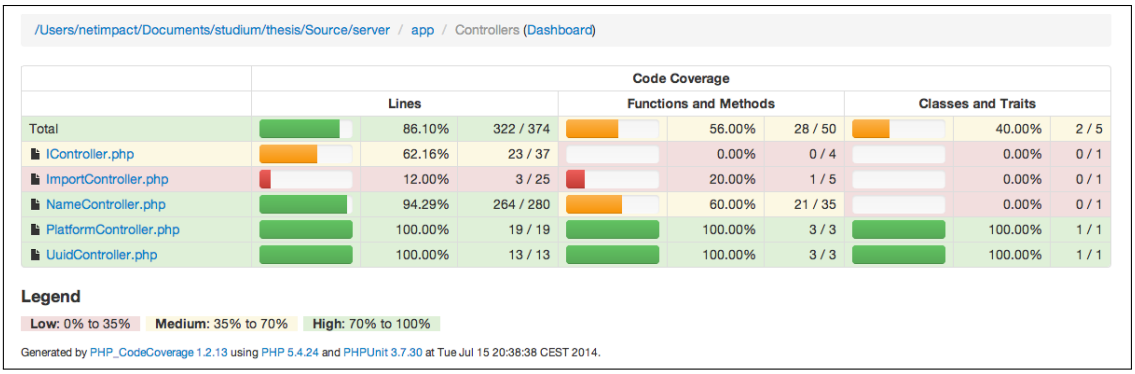

Abbildung 5.2.: PHPCodeCoverage Testreport

Die Klasse ImportController ist nur der vollständigkeithalber aufgeführt und bedarf keiner gesonderten Betrachtung, da sie nur einmalig zum Importieren der Namenslisten benötigt wurde und im Betrieb nicht benötigt wird. Deshalb können hier die Werte bzgl. der Testabdeckung vernachlässigt werden.

Um einen Test-Report für eine Test-Klasse zu genrieren wurde folgender Befehl verwendet:

```
1 $ php phpunit.phar
2         --coverage-html CodeCoverage/NameGeneration
```

```
3 NameGenerationTestCases/NameGenerationTestCases.php
```

Listing 5.6: Kommando zum Erstellen des CodeCoverage-Reports mittels PHPUnit

Um eine saubere Trennung von Test -und Produktionsmodus zu ermöglichen, wird bei der Initialisierung der Server-Komponente der jeweilige Modus (DEBUG ja oder nein) mit angegeben. Dies ist notwendig, da beim Ausführen der Tests die POST-Parameter über das Slimframework synthetisiert werden müssen.

6. Bewertung

6.1. Zielerreichung der Mustererkennung

Das Ziel, die Mustererkennung von MOBIUS zu umgehen, wurde mit dem hier vorgestellten Ansatz vollständig erreicht. Zusammenfassend lässt sich über das übergeordnete Problem bezüglich der Identifizierung in sozialen Netzwerken sagen, dass dieses teilweise gelöst wurde. Eine vollständige Lösung des übergeordneten Problems war nicht das Ziel dieser Arbeit. Das wesentliche Ziel, die Mustererkennung von MOBIUS zu umgehen, wurde nachweislich erreicht. Der Beleg dafür sind die angeführten Tests und deren erfolgreiche Umsetzung. Die Verhaltensmuster, welche zur Identifizierung von MOBIUS genutzt werden, sind somit nicht nutzbar.

Für die Client-Komponente wurde eine Testabedckung von 92.59 % ermittelt, während für den Server eine Testabedckung von 86.10 % erreicht wurde. Diese hohe Testabdeckung spricht zum einen für die Qualität der Software, sowie für den Nachweis, dass die Software erfolgreich die Verhaltensmuster, die zur Identifizierung von MOBIUS eingesetzt werden, umgehen kann. Das Konfigurieren der Komplexität von sprechenden Namen bis hin zu generierten Namen mit einer maximalen Entropie zeigt eine Bandbreite von hoher Erkennungsrate bis hin zu vollständiger Verschleierung, da die im Rahmen von MOBIUS beschriebenen Mustererkennungen fehlschlagen. Eine technische Lösung wurde somit erreicht, allerdings haben Namen mit einer maximalen Entropie den Nachteil, das Menschen sich schlecht an derartige Namen erinnern können. Die erzeugten Namen müssten in der Client-Komponente verwaltet werden und automatisch in die jeweiligen Login-Formulare eingefügt werden, um auch Benutzern mit wenig technischem Verständnis einen Mehrwert zu bieten.

Ein weitere Möglichkeit zur Verschleierung ist das Kombinieren von Namen aus unterschiedlichen Regionen, beispielsweise einem deutschen Vornamen und einem chinesischen Nachnamen. Werden diese Vor -und Nachnamen zufällig zufällig generiert und jedes Mal unterschiedliche landesspezifische Namen verwendet, so wäre die Wahrscheinlichkeit ebenfalls sehr hoch, dass die Mustererkennung fehlschlägt. Die im Rahmen dieser Arbeit aufgebaute Datenbasis erlaubt mit geringem Aufwand eine derartige Erweiterung.

6.2. Umsetzung der Anforderungen

Die beiden folgenden Tabellen zeigen, nach Anforderungsschlüssel sortiert, in wie weit die definierten Anforderungen umgesetzt worden sind.

6.2.1. Umsetzung der Client-Anforderungen

Ein Großteil der unter 4.1 aufgeführten Anforderungen wurde vollständig umgesetzt. Sollte die Umsetzung nicht vollständig sein, so wird dies in der Spalte *Bemerkung* erläutert.

Anforderungsbezeichnung	Umgesetzt	Bemerkung
CF-R-01	✓	Der Options-Bereich ist auf Deutsch, sollte aber im Zuge einer Internationalisierung, die für ein derartiges Plugin angestrebt werden sollte, mehrsprachig sein.
CF-R-02	✓	
CF-R-03	✓	Das Dialogfenster ist als JavaScript-Dialog implementiert. Besser wäre es den Dialog als HTML5 Element umzusetzen.
CF-R-04	✓	
CF-R-05	✓	
CN-FR-01	✓	
CN-FR-02	✓	
CN-FR-03	✓	
CN-FR-04	✓	
CN-FR-05	✓	
CN-FR-06	✓	
CN-FR-07	✓	
CN-FR-08	✓	
CN-FR-09	✓	
CN-FR-10	✓	
CN-FR-11	(✓)	Basic Authentication ist implementiert. Der Gebrauch von SSL ist konfiguriert, aber nicht getestet.
CN-FR-12	✓	
CN-FR-13	✓	

Die Client-Komponente wurde somit vollständig, mit geringen Einschränkungen, umgesetzt. Auf Tests im Bereich Usability wurde verzichtet.

6.2.2. Umsetzung der Server-Anforderungen

Die funktionalen und nichtfunktionalen Anforderungen aus 5.1 wurden, bis auf eine Ausnahme, vollständig umgesetzt und getestet.

Anforderungsbezeichnung	Umgesetzt	Bemerkung
S-FR-01	✓	
S-FR-02	✓	Der eingesetzte OR-Mapper, sowie die vollständige Implementierung sind auf Skalierbarkeit ausgelegt, allerdings nicht getestet.
S-FR-03	✓	
S-FR-04	✓	
S-NFR-01	✓	
S-NFR-02	✓	
S-NFR-03	✓	
S-NFR-04	✓	
S-NFR-05	✓	
S-NFR-06	✓	
S-NFR-07	✓	
S-NFR-08	✓	
S-NFR-09	✓	
S-NFR-10	✓	
S-NFR-11	✓	

Auf Last- und Performance-Tests wurde im Rahmen dieser Arbeit verzichtet.

6.3. Portierung

Die Portierung der Chrome Erweiterung in eine Erweiterung für andere Browser, wie beispielsweise Firefox, ist ohne viel Aufwand möglich, da sich die eingesetzten Technologien stark ähneln. Die Trennung von Client -und Server-Komponente bietet den großen Vorteil, dass der Server als eigenständige Komponente laufen kann. So ist es ebenso möglich eine Webseite zu erstellen, die beim Aufruf einen Namen mit der gewünschten Komplexität erzeugt.

6.4. Meta-Daten

Zwar wurde die Erkennung eines Benutzers anhand der Muster, die er bei der Vergabe eines Benutzernamens nutzt, verschleiert. Allerdings fallen beim Surfen im Internet eine Reihe von Meta-Daten an, die ebenfalls ausgewertet werden können und bereits ausgewertet werden.

Der Netzwerkverkehr wird dabei analysiert, so dass umfangreichere Lösungen nötig sind. Außerdem müssen Verhaltensregeln für Benutzer erarbeitet werden, oder solche entwickelt werden, die den Benutzer dazu erziehen nicht unnötig Datenspuren zu hinterlassen. Lösungen wie TOR, Proximax, Telex oder Flashproxy versuchen den Netzwerkverkehr zwischen Client und Server zu verschleiern. Allerdings bedeuten Fortschritte in diesem Bereich auch immer, dass die Gegenseite aufrüstet und versuchen wird Gegenmaßnahmen zu erreichen. Beispielsweise werden im TOR-Umfeld die Risiken der Exit-Nodes beschrieben, von denen es heißt, dass bei der Überwachung von ausreichend vieler solcher Nodes, Passwörter und andere sensible Daten entwendet werden können.

Darüber hinaus werden die (kommerziellen) Tracking-Verfahren perfider. Das entwickelte Canvas-Tracking, bei dem über das Zeichnen eines Bildes durch den Browser ein Fingerabdruck erzeigt wird, ist bei jedem Browser in der Standard-Einstellung möglich. Abhilfe leisten hier Browser-Erweiterungen, wie beispielsweise das von der Electronic Frontier Foundation entwickelte Privacy Badger, welches ein derartiges Browser-Verhalten verhindern soll.

6.5. Infrastruktur

Insgesamt hat sich gezeigt, dass nach dem Aufbau der Test-Infrastruktur ein gezieltes Arbeiten an der Testumsetzung und der Implementierung auf Client -und Serverseite möglich war. Alle Testfälle wurden dabei erfolgreich umgesetzt. Gleichzeitig wurde eine überdurchschnittliche Testabdeckung erreicht, die für die Qualität der entwickelten Software spricht. Die eingesetzte Infrastruktur, aus GIT als Quellcode-Verwaltung und aus einem simplen Issue-Tracker, eignet sich ausgezeichnet für Eigenentwicklung und für kollaboratives Arbeiten. Jeder Arbeitsschritt kann durch die Verknüpfung von GIT-Commits zu einem Task dokumentiert werden und erhöht so die Transparenz über den Arbeitsfortschritt.

6.6. Framework-Integration

Anfängliche Problem bei der Integration von verschiedenen Frameworks und Libraries für die Client -und Server-Komponente führten zu einem anfänglichen Mehraufwand. In Summe lässt sich aber sagen, dass dieser Mehraufwand durchaus gerechtfertigt war, da das anschließende Arbeiten mit der erarbeiteten Test-Infrastruktur nachhaltig ist. Das entwickelte Verfahren kann sich positiv sich auf Neuentwicklungen und Weiterentwicklungen auswirken, da es flexibel und modular aufgebaut ist. Außerdem kann es problemlos für das Arbeiten in Teams verwendet werden.

Mehraufwand entstand für die Server-Komponente insbesondere durch z.T. nicht kompatible ClassLoader der eingesetzten Komponenten. So verwenden Slimframework, Doctrine und PHPUnit unterschiedliche Ansätze um die benötigten Dateien zu inkludieren. Durch Kapselung und Unterscheidungen nach *DEBUG* und *PRODUCTION* wurde dieses Problem gelöst.

Für die Client-Komponente ergaben sich Schwierigkeiten durch den Einsatz von QUnit, da der testgetriebene Ansatz einer Google Chrome Erweiterung nicht dokumentiert ist und somit eine eigene Lösung entwickelt wurde. Dabei war die Ausführung der Tests im Kontext eines Chrome Tabs ein entscheidendes Hindernis, da nur so die Google Chrome API und insbesondere die Storage Funktionen getestet werden konnten. So entstand die Überlegung, die Tests innerhalb der Konfigurationsseite ausführen zu lassen.

6.7. Freigabe im Google Store

Die Extension im Google Store freizugeben, würde bedeuten, dass wenigstens Google die Implementierung, analysieren und mögliche Schwachstellen identifzieren kann. Da dies aber nur die Clientseite betreffen würde, und die Algorithmen zur Erzeugung der Namen gekapselt in der Server-Komponente verwaltet werden, ist hier kein größeres Risiko zu erwarten. Allerdings sollte vorher eine Internationalisierung der Client-Komponente erfolgen.

7. Zusammenfassung und Ausblick

7.1. Zusammenfassung

Diese Arbeit hat gezeigt, dass sich die Mustererkennung von MOBIUS umgehen lässt. Das Erstellen eines Benutzernamens mit der höchsten Komplexität erzeugt ein Namensmuster mit maximale Entropie, welches von MOBIUS nicht erkannt werden kann.

Die Entscheidung Server -und Client-Komponente zu trennen, und somit die Generierung der Namen auf dem Server zu realisieren, bietet Vorteile gegenüber einer Lösung bei der auch die Namensgenerierung innerhalb des Clients statt findet. Die Komplexität und die Zuständigkeiten sind so besser voneinander gekapselt. Zudem kann der REST-Service des Servers auch von anderen Clients angesprochen und genutzt werden. Die Wiederverwendbarkeit ist somit gegeben.

Der testgetriebene Ansatz hat sich als sinnvoll erwiesen. Probleme im Bezug auf die Implementierung wurden frühzeitig identifiziert und behoben. Das subjektive Gefühl zu jedem Zeitpunkt der Implementierung, mit jedem Iterationsschritt eine lauffähige Version sowohl client -wie auch serverseitig zu erhalten, wirkte sich positiv auf den Entwicklungsprozess aus. Nachdem die Integrationshürden von Slimframework und Doctrine behoben wurden entstand eine skalierbare Lösung mit der eine einfache Implementierung mit schlankem Quellcode auf der Serverseite möglich wurde.

Das Vorgehen, aufgrund der Analyse für Client und Server zunächst Testfälle zu entwickeln und sich auf diese zu fokussieren, erleichterte die Implementierung. Dabei spricht die hohe Testabdeckung für die Qualität der entwickelten Komponenten, und gleichzeitig dafür, dass das Problem gelöst wurde.

Zusammenfassend wurde die Problemstellung, das Umgehen der Mustererkennung mittels verhältnismäßig geringem Aufwand, erreicht. Darüber hinaus können die entwickelten Ansätze, sowie Methoden, auf Folgearbeiten übertragen werden.

7.2. Ausblick

Obgleich sich mit dem entwickelten Konzept die im Rahmen von MOBIUS vorgestellte Mustererkennung umgehen lässt, bleibt dennoch festzustellen, dass diese Arbeit bezüglich Anonymisierung nicht vollständig ist. Das Sammeln von Meta-Informationen bietet weiterhin die Möglichkeit einen Benutzer, trotz verschleiertem Benutzernamen, zu identifzieren. Außerdem muss sich ein Benutzer, sofern er anonym im Internet bewegen möchte, an viele Verhaltensregeln halten und auf Dinge, wie z.B. den Einsatz von clientseitigem JavaScript verzichten.

Der vorgestellte konzeptionelle Ansatz lässt sich auch auf andere Browser übertragen. Vorstellbar wäre eine Integration in das Firefox TOR-Bundle. Dies hätte den Vorteil, dass der enthaltene vorkonfigurierte Browser bereits sicherheitsrelevante Funktionen deaktiviert hat, die für das Identifzieren von Nutzern nötig sind. So ist beispielsweise JavaScript deaktiviert und ein AdBlocker integriert.

Die Kommunikation von Client und Server sollte mit SSL verschlüsselt sein. Dabei sollte die Übertragung der JSON-Objekte nicht in Klartext statt finden, sondern besser noch in sich selber als verschlüsselter Datenstrom. Idealerweise ist der Server dabei nur über eine .onion Top-Level-Domain erreichbar um den Standort und die Identifizierung des Hosters zu erschweren. Sofern ungewünscht der Zugriff auf den Quellcode des Servers erlangt werden kann, ist eine Verschlüsselung der PHP-Dateien mittels Zend-Guard oder Ioncube ratsam. Alternativ kann der Quellcode mit einem Obfuscator verschleiert werden, so dass zumindest ein Reverse Engineering Aufwand entsteht. Für die Datenbank gilt, dass, sofern möglich, nur verschüsselt mittlels Salts gearbeitet werden sollte. Der Server sollte so konfiguriert sein, dass keine Log-Files entstehen, die Rückschlüsse auf die jeweiligen Clients zulassen. Letztlich sollte nicht nur der Benutzername eine maximale Entropie aufzeigen, sondern ebenso, das zu dem Benutzerkonto zugehörige Passwort.

Die Trennung von Client und Server bietet die Möglichkeit die Server-Komponente auch als Stand-Alone Webseite weiter zu entwickeln. Der Aufwand, der dazu nötig wäre, ist äußerst gering. So wäre es möglich eine simple Webseite aufzusetzen, auf der letztlich nur die Komplexität des Benutzernamens ausgewählt wird. Anschließend würde ein Benutzername generiert werden.

Letztlich wäre es wünschenswert die hinterlegte Namensdatenbank mit internationalen Namen anzureichern. Eine größere Datenbasis aus gemischten Vor -und Nachnamen würde ebenfalls dazu führen, dass sämtliche Benutzermerkmale, die auf kulturellen Faktoren beruhen, nichtig wären.

A. Inhalt der CD

Dieser Arbeit wurde eine CD beigelegt, deren Verzeichnisstruktur wie folgt aufgebaut ist:

- **Dokument**: beinhaltet dieses Dokument im PDF-Format.
- **Source/client**: beinhaltet die im Rahmen dieser Arbeit entwickete Client-Komponente als Google Chrome Erweiterung sowie eine Installationsanleitung im Markdown-Format (README.md).
- **Source/server**: beinhaltet den Quellcode der Server-Kompoente, eine MySQL-Datenbank und eine Installationsanleitung im Markdown-Format (README.md). Außerdem befindet sich in dem Verzeichnis *Source/server/PHPUnit/CodeCoverage* die Auswertung der Testabedckung.

Literaturverzeichnis

[1] Beck, K.: *Test Driven Development: By Example.* Addison-Wesley Professional, 2002.

[2] Ben-Kiki, O., C. Evans und I. D. Net: *YAML.* http://www.yaml.org/spec/, 2009. [Zugriff 12.07.2014].

[3] Bergmann, S.: *PHPUnit.* http://phpunit.de/, 2014. [Zugriff 26.02.2014].

[4] Berners-Lee, T.: *RFC 1945.* http://tools.ietf.org/html/rfc1945#section-11, 2014. [Zugriff 30.06.2014].

[5] Bundesminesterium: *Bundesdatenschutzgesetz.* http://www.gesetze-im-internet.de/bdsg_1990/__3.html, 1990. [Zugriff 27.02.2014].

[6] Cake Software Foundation: *CakePHP.* http://cakephp.org/, 2014. [Zugriff 07.07.2014].

[7] Dojo Foundation: *dojo.* http://dojotoolkit.org/, 2014. [Zugriff 27.03.2014].

[8] Durand, W. und Mitwirkende: *propel.* http://propelorm.org/Propel/, 2014. [Zugriff 07.07.2014].

[9] EllisLab: *CodeIgniter.* http://ellislab.com/codeigniter, 2014. [Zugriff 07.07.2014].

[10] Google Inc.: *Google Chrome Extension.* http://developer.chrome.com/extensions, 2014. [Zugriff 28.04.2014].

[11] Google Inc.: *Google Chrome Extension Overview.* http://developer.chrome.com/extensions/overview, 2014. [Zugriff 27.03.2014].

[12] Google Inc.: *Google Chrome Extension Overview.* https://developer.chrome.com/extensions/storage, 2014. [Zugriff 30.06.2014].

[13] Handelsblatt: *Facebook-Boersengang*. http://www.handelsblatt.com/finanzen/aktien/aktien-im-fokus/ein-jahr-facebook-aktie-von-wenigen-gewinnern-und-vielen-verlierern/8225218.html, 2013. [Zugriff 27.02.2014].

[14] Hidayat, A. und Mitwirkende: *Blanket.js*. http://blanketjs.org/, 2014. [Zugriff 01.07.2014].

[15] Holland, M.: *NSA-Skandal: Von Merkels Handy, Muscular, NSA, GCHQ, BND, PRISM, Tempora und dem Supergrundrecht - was bisher geschah*. http://www.heise.de/newsticker/meldung/NSA-Skandal-Von-Merkels-Handy-Muscular-NSA-GCHQ-BND-PRISM-Tempora-und-dem-Supergru html, 2014. [Zugriff 13.04.2014].

[16] Johansen, C. und Mitwirkende: *sinon.JS*. http://sinonjs.org/, 2014. [Zugriff 27.03.2014].

[17] Kunze, M.: *Lasst es leuchten.*. c't, 12:230, 1998.

[18] Lockhart, J. und Mitwirkende: *Slim Framework*. http://www.slimframework.com/, 2013. [Zugriff 11.02.2014].

[19] Oracle: *mySQL*. https://dev.mysql.com/, 2013. [Zugriff 11.02.2014].

[20] Perito, D., C. Castelluccia, M. Kaafar und P. Manils: *How unique and traceable are usernames*. Techn. Ber., PETS, 2011.

[21] PHP-Group: *PHP*. http://www.php.net/, 2014. [Zugriff 26.02.2014].

[22] Pivotal Labs: *Jasmine*. http://jasmine.github.io/, 2013. [Zugriff 27.03.2014].

[23] Reisig, J. und Mitwirkende: *QUnit*. https://qunitjs.com/, 2008. [Zugriff 11.02.2014].

[24] Reisig, J. und Mitwirkende: *JQuery*. https://jquery.com/, 2014. [Zugriff 11.02.2014].

[25] Sadowski, D.: *PHP REST API Frameworks*. http://davss.com/tech/php-rest-api-frameworks/, 2014. [Zugriff 07.07.2014].

[26] Sjurts, I.: *Gabler Lexikon Medienwirtschaft (German Edition)*. Gabler Verlag, 2010.

[27] Slocum, J., B. Moeskau, A. Conran und R. Waters: *Sencha Ext JS*. http://www.sencha.com/products/extjs/, 2014. [Zugriff 27.03.2014].

[28] Vesterinen, K. und Mitwirkende: *Doctrine*. http://www.doctrine-project.org/, 2013. [Zugriff 11.02.2014].

[29] vks/dpa/Reuters: *Facebook-Zahlen*. http://www.spiegel.de/wirtschaft/unternehmen/quartalszahlen-facebook-verachtfacht-seinen-gewinn-a-946294.html, 2013. [Zugriff 27.02.2014].

[30] w3schools: *HTML5 Web Storage*. http://www.w3schools.com/html/html5_webstorage.asp, 2014. [Zugriff 30.06.2014].

[31] Zafarani, R. und H. Liu: *Connecting Corresponding Identities across Communities*. Techn. Ber., ICWSM, 2009.

[32] Zafarani, R. und H. Liu: *Connecting Users across Social Media Sites: a behavioral-modeling approach*. http://dl.acm.org/citation.cfm?id=2487648, 2013. [Zugriff 23.07.2014].

[33] Zend Technologies: *Zend Framework*. http://framework.zend.com/, 2014. [Zugriff 07.07.2014].

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

Hamburg, 25. Juli 2014 Alexander Simons