04-2013 | Winter | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java aktuell

Java ist nicht zu bremsen

Mobile statt Cloud
Java Enterprise Edition 7, Seite 8

Morphia, Spring Data & Co.
Java-Persistenz-Frameworks
für MongoDB, Seite 20

Starke Performance
Java ohne Schwankungen, Seite 50

Java und Oracle
Dynamische Reports innerhalb
der Datenbank, Seite 53

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

iJUG Verbund

Sonderdruck

Java-Persistenz-Frameworks für MongoDB, Seite 20

Hibernate und Extra-Lazy Initialization, Seite 38

5 Das Java-Tagebuch
Andreas Badelt, Leiter der DOAG SIG Java

8 Java Enterprise Edition 7: Mobile statt Cloud
Markus Eisele

14 Source Talk Tage 2013
Stefan Koospal

15 WebLogic-Grundlagen: Die feinen Unterschiede
Sylvie Lübeck

20 Morphia, Spring Data & Co. – Java-Persistenz-Frameworks für MongoDB
Tobias Trelle

26 Noch mehr Web-Apps mit „Play!" entwickeln
Andreas Koop

31 Flexible Suche mit Lucene
Florian Hopf

35 Automatisiertes Behavior Driven Development mit JBehave
Matthias Balke und Sebastian Laag

38 Heute mal extra faul – Hibernate und Extra-Lazy Initialization
Martin Dilger

41 Programmieren lernen mit Java
Gelesen von Jürgen Thierack

42 OAuth 2.0 – ein Standard wird erwachsen
Uwe Friedrichsen

47 „Der JCP bestimmt die Evolution der Programmiersprache …"
Interview mit Dr. Susanne Cech Previtali

49 Java ohne Schwankungen
Matthew Schuetze

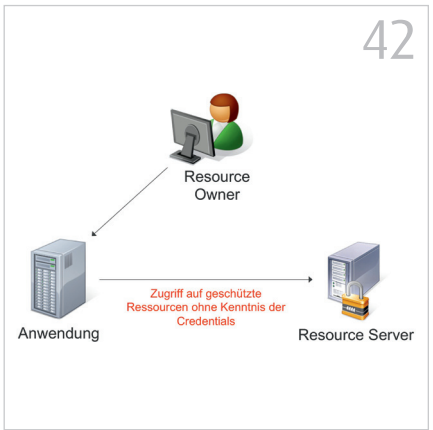
52 Dynamische Reports innerhalb von Oracle-Datenbanken
Michael Simons

56 Universelles Ein-Klick-Log-in mit WebID
Angelo Veltens

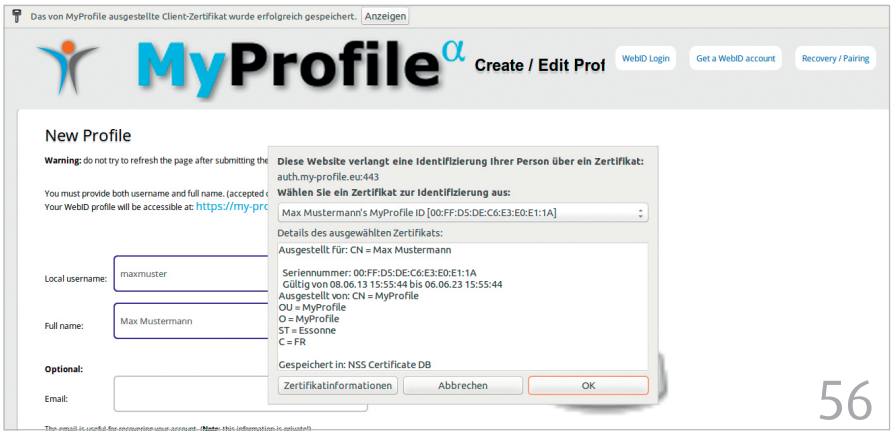
61 Leben Sie schon oder programmieren Sie noch UIs?
Jonas Helming

55 Inserenten

66 Impressum



OAuth 2.0 – ein Standard, Seite 42



Universelles Ein-Klick-Log-in mit WebID, Seite 56

Universelles Ein-Klick-Log-in mit WebID

Angelo Veltens, http://datenwissen.de

WebID bietet eine globale Identität im World Wide Web. In der letzten Ausgabe wurden die theoretischen Grundlagen dieser Technik behandelt. Dieser Artikel zeigt nun, wie sich mit WebID ein universelles Ein-Klick-Log-in für eine Web-Anwendung realisieren lässt.

Womöglich arbeitet jemand gerade an einer großartigen neuen Web-Anwendung. Doch statt sich auf deren Kern-Features konzentrieren zu können, muss er sich Gedanken um Account-Verwaltung, Log-in und Passwort-Sicherheit machen. Natürlich dürfen heutzutage auch soziale Funktionen nicht fehlen: Die Nutzerinnen und Nutzer sollen sich befreunden und Inhalte teilen können. Nicht selten kommt es vor, dass neue Web-Dienste ihr eigenes kleines Social Network implementieren, von ihren Nutzern abverlangen, abermals ein Passwort zu wählen, ein Profil auszufüllen und die Accounts ihrer Freunde zu finden. Statt das Rad neu zu erfinden, kann man WebID verwenden, um seinen Nutzern ein Ein-Klick-Log-in zu ermöglichen, das Account-Informationen und soziale Vernetzung gleich mitliefert.

Rückblick: Das Web als Social Network

Werfen wir zunächst einen kurzen Blick zurück auf Linked Data und wie damit Personen und deren Beziehungen abgebildet werden können. Wie alle Dokumente und Dinge im Linked-Data-Web werden auch Personen über einen URI identifiziert. Mithilfe des Ressource Description Frameworks (RDF) und der Friend-of-a-Friend-Ontologie (FOAF) können wir maschinenlesbare Aussagen über eine so identifizierte Person treffen. Listing 1 zeigt einen kurzen Ausschnitt aus dem FOAF-Profil des Autors.

```
<http://me.desone.org/person/aveltens#me>
        a foaf:Person;
        foaf:name "Angelo Veltens";
        foaf:homepage <http://datenwissen.de/>;
        foaf:knows <http://www.w3.org/People/Berners-Lee/card#i>.
```

Listing 1

Es enthält sowohl einfache Literale wie den Namen als auch Links zu anderen Ressourcen im Web. Diese Links sind der Schlüssel zum Aufbau eines dezentralen sozialen Netzwerks. Profile von Personen können auf unterschiedlichen Servern gehostet und trotzdem über ihren URI miteinander verlinkt werden. Bei Dokumenten ist dies seit Jahrzehnten gelebte Praxis. Linked Data wendet dieses Prinzip auch auf Daten, Dinge und Personen an. Verlinkt man die Nutzer auf diese Weise untereinander und mit den von ihnen verbreiteten Inhalten, entsteht ein globales, Server- und Dienste-übergreifendes Social Network. Das WebID-Protokoll dient dabei zur Authentifizierung der Nutzerinnen und Nutzer.

Linked Data und WebID wurden in den letzten Ausgaben ausführlich beschrieben. Wer die Artikel verpasst hat, kann sie unter [1] herunterladen. Nachfolgend wird gezeigt, wie die Authentifizierung über WebID in einer Anwendung implementiert werden kann.

Authentifizierung einer WebID

Das WebID-Protokoll kommt ohne Nutzernamen und Passwörter aus. Die Nutzer werden stattdessen über ein X.509-Zertifikat und ihren URI identifiziert. Das Zertifikat wird im Browser hinterlegt und enthält den URI des FOAF-Profils als „Subject Alternative Name". Das Profil wiederum enthält den öffentlichen RSA-Schlüssel. Um eine Nutzerin über WebID zu authentifizieren, muss ein Web-Server also zunächst eine SSL-Client-Authentifizierung durchführen und nach einem Zertifikat verlangen. Dieses Zertifikat muss gültig sein in dem Sinne, dass privater und öffentlicher Schlüssel zusammenpassen. Jedoch muss das Zertifikat nicht von einer Certificate Authority (CA) signiert oder dem Server bekannt sein. Stattdessen nutzt der Server den im Zertifikat hinterlegten URI, um die Authentifizierung und gegebenenfalls eine Autorisierung durchzuführen. Stimmt der im verlinkten Profil hinterlegte öffentliche Schlüssel mit dem des Zertifikats überein, ist nachgewiesen, dass es sich bei der Besitzerin des Zertifikats auch um die Besitzerin des Profils handelt.

Um eine Authentifizierung mit WebID zu ermöglichen, können die genannten Schritte manuell implementiert werden. Nachfolgend werden jedoch zwei alternative Wege aufgezeigt, die auf bestehende Lösungen setzen und die Sache deutlich vereinfachen. Die erste Variante delegiert die Authentifizierung an einen externen Dienst. Die zweite nutzt das Apache-Modul „mod_authn_webid", um die Authentifizierung auf einem eigenen Server durchzuführen. Wer sich dennoch im Detail mit einer eigenen Implementierung der WebID-Spezifikation auseinandersetzen möchte, findet sie unter [2].

Eine WebID erstellen

Bevor wir uns an die eigentliche Implementierung machen, wollen wir uns zunächst selbst eine WebID erstellen. Schließlich soll die Authentifizierung auch getestet werden. Am einfachsten geht dies über einen Profile-Hoster wie „my-profile.eu". Unter [3] gibt man lediglich einen frei wählbaren Namen und einen eindeutigen lokalen Benutzer-Namen an. Letzterer wird

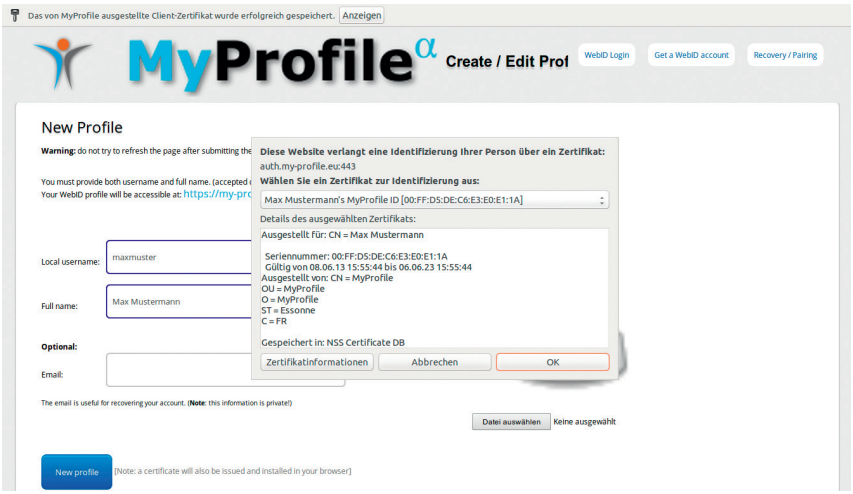


Abbildung 1: WebID-Log-in bei my-profile.eu

Teil des URI. Nach einem Klick auf „New Profile" generiert der Browser ein Zertifikat und speichert es in der Zertifikatsverwaltung ab. Anschließend kann man sich mit einem Klick auf „WebID Log-in" einloggen (siehe Abbildung 1).

Wer sein Profil lieber selbst hosten möchte, schreibt einfach eine RDF/XML- oder Turtle-Datei und hinterlegt sie auf seinem Server. Wichtig dabei ist, dass der persönliche URI sich von dem des Dokuments unterscheidet, also beispielsweise durch ein „#me" ergänzt wird. Ein Zertifikat lässt sich beispielsweise über das Kommandozeilen-Tool „openssl" erzeugen. Der öffentliche Schlüssel wird anschließend, wie in der letzten Ausgabe beschrieben, in seinem Profil eintragen. Unter [4] gibt es ein fertiges Shell-Skript, das ein WebID-taugliches Zertifikat generiert und die zugehörigen RDF-Tripel ausgibt.

Authentifizierung delegieren

Die Authentifizierung an einen externen Dienst zu delegieren, hat den Vorteil, dass lediglich das API des Dienstes implementiert werden muss und nicht das komplette WebID-Protokoll. Der Nachteil ist, dass zusätzliche HTTP-Roundtrips entstehen und dem Dienst vertraut werden muss. Für den Einstieg ist dies jedoch die einfachste Variante. Wir werden im Folgenden den Dienst „foafssl.org" für die Authentifizierung heranziehen. Als Web-Framework kommt Grails zum Einsatz, die Prinzipien lassen sich jedoch leicht auf andere Web-Frameworks übertragen. Die Authentifizierung der Nutzer erfolgt in drei Schritten:

1. Weiterleitung des Nutzers zu „foafssl.org"
2. Auswertung der Antwort von „foafssl.org"
3. Verknüpfung des WebID-Nutzers mit der eigenen Anwendung

Im ersten Schritt werden die Besucher der Website also zu „foafssl.org" weitergeleitet – und zwar zu „https://foafssl.org/srv/idp?rs=<Antwort-URI>". Dabei ist „<Antwort-URI>" ein bestimmter URI der Web-Anwendung, der die Antwort von „foafssl.org" entgegennehmen und weiterverarbeiten kann. Die Nutzer werden bei Aufruf von „/webId/login" weitergeleitet und die Antwort von „foafssl.org" unter „/webId/verify" erwartet. Dazu legt man einen „WebIdController" mit den Actions „login" und „verify" an (siehe Listing 2).

Mittels „redirect()" wird bei Grails eine Weiterleitung eingeleitet. Den für den „rs"-Parameter benötigten absoluten URI erzeugen wir zuvor mit „createLink()". Er verweist auf die „verify"-Action. Die Antwort von „foafssl.org" erfolgt durch eine Weiterleitung an diesen „verify"-URI, inklusive der folgenden drei Parameter:

- *webid*
  Die WebID des authentifizierten Users
- *ts*
  Ein Zeitstempel, wann die Authentifizierung durchgeführt wurde
- *sig*
  Die Signatur von „foafssl.org"

Um sicherzustellen, dass die angegebene WebID tatsächlich korrekt authentifiziert wurde, müssen sowohl die Signatur als auch der Zeitstempel überprüft werden. Bei der Signatur handelt es sich um den uns bekannten „verify"-URI, ergänzt um die oben genannten Parameter „webid" und „ts", der mit dem RSA-Schlüssel von „foafssl.org" signiert wurde. Mit dem öffentlichen Schlüssel von „foafssl.org" können wir diese Signatur überprüfen. Den Schlüssel findet man ebenso wie die Dokumentation des Dienstes unter [5]. Java bietet alles, was wir zur Überprüfung der Signatur benötigen (siehe Listing 3).

In den ersten fünf Zeilen wird eine Public-Key-Instanz aus den öffentlichen Daten „MODULUS" und „PUBLIC_EXPONENT" erzeugt. In der sechsten Zeile rekonstruieren wir den URI, den „foafssl.org" signiert haben sollte. Mithilfe eines Signatur-Objekts prüfen wir anschließend, ob dies tatsächlich die Daten sind, die von „foafssl.org" signiert wurden. Dabei ist zu beachten, dass die Signatur „Base64-URL"-encodiert (im Gegensatz zur gewöhnlichen Base64-Encodierung werden hierbei die Zeichen „+" und „/" durch „–" und „_" ersetzt) ist und zunächst decodiert werden muss. Dies erledigt „Base64.decodeBase64()" aus Apache Commons für uns.

```
class WebIdController {
        def login() {
                def rs = createLink (
                        action: 'verify',
                        absolute: true
                ).encodeAsURL()
                redirect(url: "https://foafssl.org/srv/idp?rs=${rs}")
        }
        def verify() {
                // Signatur und Timestamp prüfen (Listings 3 und 4)
        }
}
```

Listing 2

Um Replay-Attacken zu erschweren, ist auch der Zeitstempel unbedingt zu überprüfen. Gelangt ein Angreifer zum Beispiel an den „verify"-URI von WebID-Nutzerin „Alice", könnte er sich damit jederzeit als „Alice" bei unserem Dienst einloggen. Der „verify"-URI sollte daher eine möglichst kurze Zeitspanne gültig sein. Allerdings darf diese Zeitspanne auch nicht zu kurz sein, da sonst gültige Requests abgewiesen werden, weil sie etwas länger dauern oder die Zeiten des „foafssl.org"-Servers und des von uns verwendeten Web-Servers zu stark voneinander abweichen. Hat man sich für eine angemessene Lebensdauer entschieden, fällt die Implementierung leicht (siehe Listing 4).

Der von „foafssl.org" übermittelte Zeitstempel entspricht dem Datumsformat „yyyy-MM-dd'T'HH:mm:ssZ" und kann entsprechend geparst werden. Anschließend muss er nur noch mit der aktuellen Uhrzeit verglichen werden. Ist die Differenz größer als die tolerierte Lebenszeit, muss die Authentifizierung scheitern.

Nachdem so sichergestellt wurde, dass die Antwort von „foafssl.org" sowohl korrekt als auch aktuell ist, kann die WebID beispielsweise genutzt werden, um den zugehörigen lokalen Account zu finden oder gegebenenfalls einen solchen anzulegen. Anschließend kann die Anwendung wie gewohnt arbeiten, als ob der Nutzer sich mit Benutzername und Passwort angemeldet hätte (siehe Listing 5 und Abbildung 2). Eine große Stärke von WebID liegt jedoch auch darin, dass über den URI weitere Informationen nachgeladen werden können, wie wir später noch sehen werden.

```
BigInteger MODULUS = new BigInteger("9093ac0285..." , 16);
BigInteger PUBLIC_EXPONENT = new BigInteger("65537", 10);
def publicKeySpec = new RSAPublicKeySpec(MODULUS, PUBLIC_EXPONENT);
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
PublicKey publicKey = keyFactory.generatePublic(publicKeySpec);
byte[ ] data = "$verifyUrl?webid=${URLEncoder.encode(webId)}&ts=${URLEncoder.
encode(timestamp)}".getBytes()
Signature sig = Signature.getInstance("SHA1withRSA");
sig.initVerify(publicKey);
sig.update(data);
return sig.verify(Base64.decodeBase64(signatureBase64));
```

Listing 3

```
Date verifiedDate =
        new SimpleDateFormat(TIMESTAMP_FORMAT).parse(timestamp)
def differenceInMillis =
        new Date().getTime() - verifiedDate.getTime()
return (differenceInMillis / 1000) <= TIMESTAMP_LIFESPAN
```

Listing 4

```
session.user = webIdAccountService.loadOrCreateUserAccount(webId)
flash.message = "Successfully logged in with WebID $webId"
redirect (controller:'myAccount')
```

Listing 5

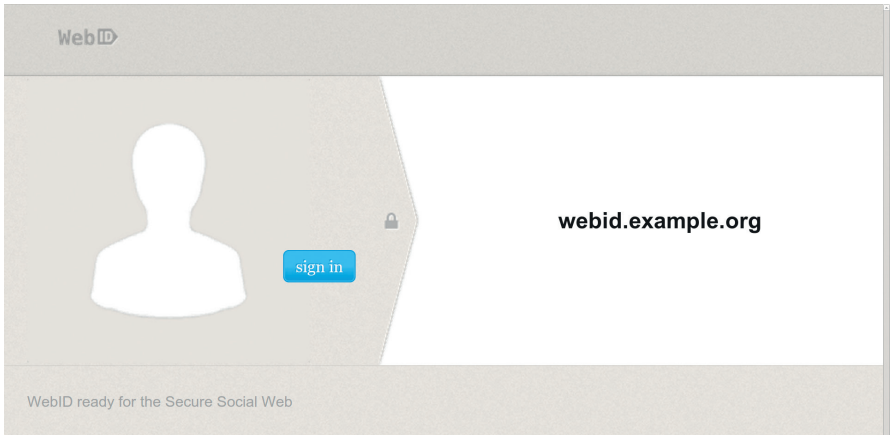


Abbildung 2: Externe WebID-Authentifizierung über „foafssl.org"

Authentifizierung selbst gemacht

Die Delegation der Authentifizierung an einen externen Dienst wie „foafssl.org" ist einfach zu realisieren, bringt jedoch auch Nachteile mit sich. Es sind mehrere HTTP-Weiterleitungen erforderlich, was den Prozess in die Länge zieht und noch nicht viel vom versprochenen Ein-Klick-Log-in erahnen lässt. Weiterhin ist es nicht immer angemessen, einem externen Dienst die Authentifizierung anzuvertrauen. Dieses Problem steht bereits der Verbreitung von OpenID im Wege. Bis heute sind viele OpenID-Provider trotz des offenen Standards nicht miteinander kompatibel, da sie sich gegenseitig nicht vertrauen. Bei WebID tritt dieses Problem nicht auf, da jeder Dienst die Authentifizierung selbst durchführen kann.

Mithilfe des Apache-Moduls „mod_authn_webid" lässt sich die Authentifizierung leicht auf dem eigenen Server konfigurieren. In folgendem Beispiel-Szenario läuft eine Grails-Anwendung auf einem Apache Tomcat. Ein Apache-httpd-Server ist mittels „mod_jk" als Reverse-Proxy vor diesen geschaltet. Die Authentifizierung wird der Apache-httpd übernehmen und das Ergebnis an Apache Tomcat weiterreichen.

Das Modul „mod_authn_webid" muss zunächst aus den Sourcen [6] kompiliert werden, was jedoch problemlos gelingen sollte, wenn die in der README-Datei genannten Voraussetzungen erfüllt sind. Listing 6 zeigt,

```
apt-get install librdf0-dev
apt-get install openssl
autoconf
./configure
make install
```

Listing 6

wie der Autor das Modul auf einem Ubuntu-System installieren konnte. Anschließend muss das Modul geladen und der Apache neu gestartet werden (siehe Listing 7).

Sobald das Modul verfügbar ist, kann ein virtueller Host für die Grails-Anwendung eingerichtet werden. Angenommen, die Anwendung soll über die Domain „webid.example.org" erreichbar sein und unter „/login" ein WebID-Log-in ermöglichen. Da WebID auf SSL aufsetzt, muss mindestens dieser Log-in-URI über „HTTPS" abgesichert sein. Um dies sicherzustellen, richten wir für diesen Pfad einen Redirect zur HTTPS-Variante ein (siehe Listing 8). Die Konfiguration für die HTTPS-Variante ist in Listing 9 ersichtlich.

```
echo 'LoadModule authn_webid_module /usr/lib/apache2/modules/mod_auth_webid.so' >> /etc/
apache2/mods-available/authn_webid.load
a2enmod authn_webid
service apache2 restart
```

Listing 7

```
<virtualhost *:80>
        ServerName webid.example.org
        Redirect permanent /login https://webid.example.org/login
        JkMount /* worker1
</virtualhost>
```

Listing 8

```
<virtualhost *:443>
        ServerName webid.example.org
        SSLEngine On
        SSLCertificateFile /etc/apache2/ssl/server.pem
        SSLOptions +StdEnvVars +ExportCertData
        SSLVerifyClient none
        <Location /login>
                SSLVerifyClient optional_no_ca
                SSLVerifyDepth 1
                AuthType WebID
                Require valid-user
        </Location>
        JkMount /* worker1
</virtualhost>
```

Listing 9

```
def login () {
        String webId = request.getRemoteUser ()
        if (webId){
                loadOrCreateUserAccount(webId)
                flash.message = "Successfully logged in with WebID $webId"
                redirect (controller: 'myAccount')
        } else {
                flash.message = 'Login failed.'
                session.user = null
                redirect (controller: 'home')
        }
}
```

Listing 10

```
def rdfLoader = new JenaRdfLoader ()
RdfResource person = rdfLoader.loadResource(webId)
String name = person(foaf.name)
String imgUri = person(foaf.img).uri
String cityUri = person(foaf.basedNear).uri
RdfResource city = rdfLoader.loadResource(cityUri)
String cityName = city(geo.name)
```

Listing 11

Die Option „SSLOptions +StdEnvVars +ExportCertData" ist wichtig, damit die Zertifikat-Informationen für Apache Tomcat und die Grails-Anwendung verfügbar sind. Über „ SSLVerifyClient none" wird grundsätzlich die Notwendigkeit zur Authentifizierung ausgeschaltet. Lediglich für den Log-in-URI wird im Location-Abschnitt die WebID-Authentifizierung eingeschaltet. „SSLVerifyClient optional_no_ca" stellt dabei sicher, dass alle gültige Zertifikate akzeptiert werden, unabhängig davon, ob sie von einer CA signiert und dem Server bekannt sind.

Mit „AuthType WebID" kommt das soeben installierte Modul ins Spiel. Der Browser fragt nun beim Aufruf des Log-in-URI nach einem Zertifikat und prüft, ob es sich um eine gültige WebID handelt. Falls eine Nutzerin erfolgreich authentifiziert werden konnte, steht deren URI anschließend in der Umgebungsvariable „REMOTE_USER".

In der Grails-Anwendung lässt sich diese bequem über „request.getRemoteUser()" abfragen. Ist sie gesetzt, liegt ein authentifizierter WebID-Nutzer vor, der beispielsweise mit einem lokalen Account verbunden werden kann. Ist die Variable leer, ist das Log-in fehlgeschlagen (siehe Listing 10).

Das war´s dann. Ein Klick auf den Log-in-Link führt nun dazu, dass der Browser um die Auswahl eines Zertifikats bittet. Nachdem das WebID-Zertifikat ausgewählt ist, ist man eingeloggt. Da sich der Browser das ausgewählte Zertifikat merkt, ist es nun bei allen Diensten im Web, die WebID unterstützen, möglich, sich mit einem einzigen Klick anzumelden.

Daten bitte

Über den WebID-URI können wir nicht nur Nutzer global identifizieren, sondern auch zusätzliche Informationen laden. Denn schließlich verbirgt sich dahinter ein gegebenenfalls sehr komplexes, maschinenlesbares FOAF-Profil. In der vorletzten Ausgabe haben wir mithilfe der Bibliothek

„groovyrdf“ [7] bereits Daten über Hotels aus dem Web-of-Data geladen. Genauso können wir auch mit Informationen über Personen verfahren (siehe Listing 11).

Neben einfachen Literalen wie dem Namen lassen sich auch verlinkte Ressourcen abrufen und Informationen über diese nachladen. In dem Beispiel-Listing wird der Link zum Ort der Person verfolgt und dessen Name abgerufen. Da sich die Profil-Informationen beliebig zusammensetzen können und nicht unter Kontrolle der Anwendung liegen, sollte die Abfrage der Daten möglichst tolerant erfolgen und fehlende Informationen sollten verschmerzbar sein. Möglicherweise enthält ein Profil beispielsweise gar kein „foaf:basedNear“-Prädikat oder dieses enthält Geo-Koordinaten, anstatt auf eine andere Ressource zu verlinken. Die Profil-Informationen sind außerdem ebenso vorsichtig zu behandeln wie gewöhnliche Nutzer-Eingaben. Wird der Name einer Person achtlos in die HTML-Seite einbettet, ist die Anwendung zum Beispiel für Cross-Site-Scripting (XSS) anfällig.

Um tatsächlich ein globales, dezentrales Social-Network aufzubauen, ist es sehr zu empfehlen, dass man die URIs seiner Nutzer mit den Inhalten der Anwendung verlinkt. Ermöglicht es die Anwendung beispielsweise, Nutzer-Inhalte zu „liken“, sollten man diese Information auch selbst wieder als RDF-Tripel veröffentlichen oder ein semantisches Pingback [8] senden, um das „Social Web of Data“ zu erweitern. Veröffentlicht man alle Inhalte auch als Linked Data, haben andere Dienste die Möglichkeit, auf sie zu verlinken.

Fazit und Ausblick

Mithilfe von WebID bietet man Nutzern eine bequeme Log-in-Möglichkeit mit einem bestehenden und gegebenenfalls sogar selbst-gehosteten Profil. Selbst wenn die Nutzer noch keine WebID besitzen, kann man sie zur Profil-Anlage und -Verwaltung leicht an einen externen Dienst wie „myprofile.eu“ verweisen und dennoch die Authentifizierung nicht aus der Hand geben.

Es ist auch problemlos möglich, WebID als Ergänzung zu etablierten Log-in-Möglichkeiten anzubieten. Mit dem Aufkommen neuer WebID-tauglicher Dienste wird die Verbreitung von WebID weiter steigen und sich der Nutzen für die Besitzer einer WebID und die Betreiber von Diensten vervielfachen. Je mehr Dienste und Nutzer nach diesem Prinzip das Netz verwenden, desto stärker wird die Verlinkung von Daten und sozialen Interaktionen. Letztlich wird die Summe an kleinen, vernetzten Diensten im Social-Web die Möglichkeiten eines zentralisierten sozialen Netzwerks weit übersteigen.

Referenzen

[1] https://datenwissen.de/hintergrundinfos/zeitschriftenartikel/
[2] http://www.w3.org/2005/Incubator/webid/spec/
[3] https://my-profile.eu/profile
[4] https://gist.github.com/njh/2432427
[5] https://foafssl.org/srv/idp
[6] https://github.com/linkeddata/mod_authn_webid
[7] http://angelo-v.github.io/groovyrdf/
[8] http://www.w3.org/wiki/Pingback#Semantic_Pingback

Angelo Veltens
angelo.veltens@online.de
http://datenwissen.de

Angelo Veltens studierte Angewandte Informatik an der Dualen Hochschule Baden-Württemberg Karlsruhe und befasste sich in Studienarbeiten und Abschlussarbeit mit Linked Data und semantischem Wissensmanagement. Heute ist er als Software-Entwickler in Braunschweig tätig, arbeitet in seiner Freizeit am „Social Web of Data“ und referiert auf Konferenzen zu diesem Thema.

Schulungstag

- Wissensvertiefung für Oracle-Anwender
- Mit ausgewählten Schulungspartnern
- Von Experten für Experten

22. November 2013

Buchen Sie 1 Tag englischsprachige Intensiv-Schulung zum Thema JavaServer Faces (JSF 2.2), dem Standard Web-Applikations-Framework für Java EE.
mit **Ed Burns**

iJUG Verbund
www.ijug.eu

DOAG Deutsche ORACLE-Anwendergruppe e.V.
www.doag.org

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG *News* und vier Ausgaben im Jahr Business News zusammen für 70 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN
0700 11 36 24 39

ODER BESTELLEN SIE ONLINE
go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr
☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:
Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.