Aalto University

Bachelor’s Thesis

Analysis and Implementation of a SSO Solution for Several Web Portal

Author: Xavi Magrinyà
Supervisor: Professor Petri Vuorimaa

A Thesis submitted in fulfilment of the requirements for the degree of BSc. in Computer Science in the

Department of Media Technology

August 2012
Declaration of Authorship

I, Xavi MAGRINYÀ, declare that this Thesis titled, 'Analysis and Implementation of a SSO Solution for Several Web Portal' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 

 undersign
“Education must provide the opportunities for self-fulfillment; it can at best provide a rich and challenging environment for the individual to explore, in his own way.”

Noam Chomsky
When we talk about Single Sign-On (SSO), there is much new research done during the last years. The main goal of SSO is to provide a unique way to authenticate users along many websites. Due to the structure of the Active Life and the other partners portals, a SSO is needed to access to the data by the easiest way. Moreover, the portal is targeted also to elderly people who should find as easy as possible to authenticate within the portal. This is the reason why the SSO needs to be properly implemented. The theoretical part of this Thesis is focused in the existing SSO systems, which one fits better into the system and how do they work. After this, the implementation of the selected SSO is explained and discussed, as well as linked with the theory. At the end of the Thesis, the conclusions and difficulties found are reported.
Acknowledgements

Thanks to everybody that has collaborated in this project, especially to my supervisor Petri Vuorimaa for giving me the opportunity of doing this Thesis.
## Contents

Declaration of Authorship ........................................... i

Abstract ........................................................................ iii

Acknowledgements ........................................................... iv

List of Figures ..................................................................... vii

Abbreviations ..................................................................... viii

1 Introduction .................................................................... 1

  1.1 Overview .................................................................... 1

  1.2 Research Problem .......................................................... 1

    1.2.1 The current implementation ...................................... 2

    1.2.2 Reasons to improve the SSO ...................................... 2

  1.3 The research question ..................................................... 3

  1.4 Structure of the Thesis .................................................... 4

2 Single Sign-On Solutions .................................................. 5

  2.1 Overview .................................................................... 5

  2.2 Social Security Number Field ......................................... 5

  2.3 Shibboleth .................................................................... 6

    2.3.1 The components of Shibboleth ................................. 6

    2.3.2 How does Shibboleth work? ................................. 6

  2.4 OpenID ...................................................................... 7

    2.4.1 OpenID Working System ....................................... 7

  2.5 OAuth ........................................................................ 8

    2.5.1 OAuth Protocol ................................................... 10

      2.5.1.1 OAuth Overview .......................................... 10

      2.5.1.2 OAuth Specification Details ......................... 12

3 Liferay ......................................................................... 14

  3.1 Introduction to Liferay .................................................. 14

  3.2 Liferay Portal Development ......................................... 14

    3.2.1 Portlets ................................................................. 15
4 Solution analysis

4.1 Requirements of the solution .................................................. 17
4.2 Analysis ...................................................................................... 18
  4.2.1 Social Security Number .......................................................... 18
  4.2.2 Shibboleth .............................................................................. 19
  4.2.3 OpenID ................................................................................. 19
  4.2.4 OAuth ...................................................................................... 20
4.3 Comparison of the SSO Systems .................................................. 21
4.4 Selected SSO System ................................................................. 22

5 Implementation

5.1 How to implement OpenID in Liferay .......................................... 24
5.2 The Client ................................................................................... 24
  5.2.1 Using OpenID Java Library .................................................... 25
  5.2.2 Forms and JSP ....................................................................... 28
  5.2.3 The Database .......................................................................... 30
5.3 The OpenID Provider ................................................................. 31

6 Conclusions

A OpenID Core Code

B JSP Login Form Code

References
List of Figures

1.1 Database Implementation ........................................... 2
2.1 Shibboleth ............................................................. 7
2.2 OpenID logo ........................................................... 8
2.3 OpenID schema ....................................................... 8
2.4 OAuth ................................................................ 11
3.1 Portlet ................................................................. 16
4.1 Schema ................................................................. 20
4.2 Table ................................................................ 21
5.1 Liferay database ..................................................... 30
5.2 Users table and OpenID field ..................................... 31
Abbreviations

ALH  Active Life Home
AOL  America Online
API  Application Programming Interface
BBC  British Broadcasting Corporation
CMF  Content Management Framework
CMS  Content Management System
CSS  Cascading Style Sheets
EE   Enterprise Edition
HTML HyperText Markup Language
HTTP HyperText Transfer Protocol
HTTPS HyperText Transfer Protocol Secure
IBM  International Business Machines
IP   Identity Provider
J2EE Java 2 Enterprise Edition
JOIDS Java OpenID Server
JSON JavaScript Object Notation
JSP  JavaServer Pages
OP   OpenID Provider
PHP  Hypertext Preprocessor
RDBMS Relational Database Management System
RFC  Request for Comments
RO   Resource Owner
RP   Relying Party
RS   Resource Server
RSS  Rich Site Summary
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign-on</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>WAR</td>
<td>Web application ARchive</td>
</tr>
<tr>
<td>WAYF</td>
<td>Where Are You From</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XRI</td>
<td>Extensible Resource Identifier</td>
</tr>
</tbody>
</table>
Dedicated to my family for the support received.
Chapter 1

Introduction

1.1 Overview

The main goal of this Thesis is to implement a Single Sign-on for a Web Portal. In other words, we wanted to find a solution to implement a unique web authentication system for different websites without having to login again every time that you want to access to each of the portal’s services.

In this case, the Portal will be the website of Active Life Home. This website provides different solutions for the health care of elderly people, merging different devices such as medicine dispensers, alarm bracelets or movement sensors into one solution. Thus, we need to integrate different services offered by participating companies into one web solution, using their own web interface. In order to use these services, we need to login into their portals using a unique solution to access to the data of the logged user.

This Thesis is about how can we do this, what kind of solutions exist to share a SSO solution, discuss about which solution is the best and how to implement this solutions into an existing portal. Also this solution has to be secure, trustable, non-depending of third parties and easy to implement to make as easy as possible the developing process of the other partner’s websites.

1.2 Research Problem

The current portal implementation does not have a good login solution to integrate all the partners services. There are many reasons why the existing implementation is not a good solution. We are going to discuss them in more detail. After this, will be defined the research question.
1.2.1 The current implementation

In the current implementation, each partner’s website has a different login system. So in order to access to the different login systems, each account of each partner is linked to the user’s database of the Active Life Home portal.

As we can see in Figure 1, each user and password of each partner is represented in the database with a specific field. Furthermore, this provisional solution for the login system does not include any password protection for the partners passwords and they are showed in the database as plain text without any kind of encryption.

1.2.2 Reasons to improve the SSO

The actual SSO system has many shortages. As it was a provisional solution to login into other partner’s websites, it still has to be improved. One reason is that this solution is not scalable. The database has to be modified every time there’s a new partner by adding two new fields for that specific partner’s login. This makes the procedure of linking the accounts a tough work every time there’s a new partner because all the users have to be introduced manually into the database.

There are also other problems. The database is not centralised, so the data is divided in different databases. That means that there has to be coordination within the databases. This coordination does not exist at the moment and it is difficult to implement. This lack of coordination carries other problems. For example, every time that the username or the password of a partner’s website is modified it has to be modified again in the main
Chapter 1. Introduction

database in order to keep up to date the SSO. Implementing a way to do it automatically can be a waste of time, and the aim is to make it easier.

Another important thing to take into count is that the user data for other portals is stored as plain text, which is totally unsecure. An easy attack can be used to take the user’s login information without any kind of encryption or protection. That can make other partners realise that the portal is not reliable anymore and stop using the existing login system.

1.3 The research question

As it is discussed above, the existing SSO system is not appropriate for the purpose of the portal. So this Thesis tries to find a new and better way to improve the existing solution. First, we need to understand which are the needs of the new SSO, how to solve the problems mentioned above and compare the different solutions. Once we decide which SSO is better, we can focus on the implementation and the way it is integrated with the current system.

So the main research question is: which is the best SSO solution for the Web Portal and how can it be implemented? To solve this, I’m going to divide the question into two sub-questions that I’m going to explain. I’m going to refer to these questions as Q1 and Q2.

Q1: Which is the most suitable SSO solution for the Web Portal?

As discussed above, the current solution is far from being a good solution for a SSO. It is necessary to analyse different solutions and which one of them is the most suitable for our problem. There are many SSO solutions, most of them open source but we will discuss other types of non-open source also. Taking example of other websites implementations is a good way to find which possible solutions can be used and which ones are the most common used. These solutions are going to be explained later in more detail.

Q2: How to implement the selected SSO solution?

The Web Portal that we are talking about is using Liferay. Liferay is a Content Management System (CMS) implemented in Java EE, so the solution found in Q1 has to be integrated in this CMS using Java EE. This is a good point, because most of the SSO open source solutions have implemented libraries in Java. Also the non-open source solutions can be easily implemented into the Web Portal.
1.4 Structure of the Thesis

The structure of this Thesis is the following one. First it starts explaining some theory about the proposed SSO systems, a brief introduction and how it is implemented. This is explained in Chapter 2, while Chapter 3 is focused on Liferay. This is important to understand because the implementation of the portal is done using Liferay CMF. Further in Chapter 4, the SSO systems are going to be analysed, taking the benefits and drawbacks of each one and comparing them in detail. After the candidates are compared, we will show which option is the one that best fits the needs of the portal. Chapter 5 will show how to implement OpenID within Liferay, and how to integrate it in the current solution. Finally, in Chapter 6 there are the conclusions of the Thesis.
Chapter 2

Single Sign-On Solutions

2.1 Overview

There are some possible solutions to solve the research question. Not all of them are valid for the purpose of the Web Portal. These are the proposed solutions. The following chapter is going to explain in more detail the main proposed solutions, including technical specifications of how it works.

2.2 Social Security Number Field

This was one of the first solutions given. The main point is to add a common field in the database to all the partners with the Social Security number. The idea is to use this identification number to login into the other partners websites.

The procedure is the following. Each user has his own Social Security number in Finland. This number is linked into the user information on the database, so there is a mandatory field in the database with the Social Security number of each user independently of their role. When the user wants to login into a partners website, this number is sent to the partners website to recognize the user. The partners website searches in its database the Social Security number and identifies the user. Then the selected user is logged in.
2.3 Shibboleth

Shibboleth is a software project that aims to give an open solution to the user authentication and authorisation and the information exchange between different organizations. Internet2, a United States non-profit network consortium led by many important members of communities, industry and government, promotes this project. The project started in 2000 to solve problems between organizations with different authentication and authorisation infrastructures. Shibboleth's implementation is based on SAML. SAML, acronym for Security Assertion Markup Language, is an XML-based standard to exchange authentication and authorisation data. Shibboleth 2.0 uses the standards of SAML 2.0, and is the current version and the one that we will talk about.

2.3.1 The components of Shibboleth

To explain in more detail how Shibboleth works, we first need a description of the components that take part in the process. There are three components: the Identity Provider, the Service Provider and the Where Are You From server.

The IP is in charge of giving assertions of authentication or some attributes to the SP. The exchange of attributes means that the SP does not have to know the whole identity of the user, he might want just to know some attributes that are necessary (e.g., the age of the user) dismissing unnecessary information (e.g., the organisation that it belongs).

The SP manages protected resources depending on the information received by the IP. It asks the IP for the authentication or attributes of a user and process the answer, providing the information according to the answer of the IP, protecting the rest of the non-authorised information.

The WAYF server is used to select the IP. It is a middle step between the SP and the IP. This server does not always exist, and it is possible to make that the selected IP is never asked again using cookies.

2.3.2 How does Shibboleth work?

Once we know the main components in the authentication and authorisation procedure of Shibboleth we can explain how does it work, according to the next figure. The procedure is the following one.

The user requests a page (1) protected with Shibboleth to the SP. The SP looks if the user is authenticated and if its not, redirects the user to the WAYF server (2). The
WAYF shows a list of possible IPs and the user selects its own provider and redirects the page to the selected IPs page (3). In a practical case, the WAYF server not always appears. It might happen that the user has the information on cookies or simply there is not WAYF server and redirects to an IP directly. Once in the IP’s page, the user authenticates itself using a form (4). The IP sends to the SP a SAML assertion with the authentication information saying whether it is or not authenticated (5). After the authentication process is done, the SP asks to the IP for the attributes needed (6) and the IP decides if it is possible to give the attributes to the SP and returns the requested information (7). Then the SP displays the requested page with the protected information to the user (8). If the user requests for another page and it is already authenticated and authorised, the SP does not ask to the IP for the veracity of the authentication every time the user makes a request. This procedure is done only once per session.

2.4 OpenID

OpenID is a decentralised open-source project to manage authentication and authorisation. Lots of websites and large companies such as Google, Yahoo, PayPal, IBM, BBC, AOL, VeriSign or MySpace use this authentication method. OpenID original protocol was developed in May 2005 and gained popularity due to decentralisation and the freedom of being an OpenID Provider without the approval of any organisation. At the end of 2009, there were over one billion OpenID accounts in the Internet and approximately 9 million websites were integrating it.
The way that OpenID works is really simple to explain. As in Shibboleth there is a SP and an IP, but we are going to call them Relying Party and OpenID Provider. However, in OpenID there is no WAYF server because it is not necessary. Now we are going to explain in more detail how it works.

2.4.1 OpenID Working System

As said, there are three major components in the authentication process: the user (linked to the User-agent, such like browsers), the RP and the OP. Figure 4 shows in easy steps how does OpenID work.

1. First, the user requests to login into a RP’s webpage that implements OpenID.
2. The RP returns a login form to the user to fill with its OpenID information. This form has no password field. It is just to input the OpenID Identifier.

3. The user introduces its own OpenID account and sends it back to the RP. The OpenID Identifier introduced is an http or https URL or an XRI. In the form sent the attribute name should have the value openid_identifier in order to support browser extensions and other software that supports OpenID.

4. The RP looks for the introduced OpenID account and sends to the OP an authentication request for the user and redirects the user to the authentication page of the OP. The authentication request is an indirect request, which means that it uses one of the two methods: HTTP redirect or HTTP form submission, and require that the sender knows the recipient URL and the recipient URL expect indirect messages. The authentication request must include a list of parameters like openid.ns or openid.mode. There are also some optional parameters like the OpenID-Local Identifier or the Return URL.

5. The user authenticates himself into the OP. There is no documentation about the way that an OP has to authenticate with the user, so this point depends on the OP.

6. Then the OP returns an assertion with the information of the authentication result, saying if the authentication is approved sending a positive assertion or if it is denied sending a message saying that the authentication failed. The positive assertion must include some parameters like:

   (a) openid.ns: with the specs of the OpenID protocol
   (b) openid.mode: set to id_res
   (c) openid.op_endpoint with the OP endpoint URL
   (d) openid.return_to: with the return URL
   (e) openid.response_nonce: with a unique string for the authentication response
   (f) openid.assoc_handle: with the handle for the association that was used to sign this assertion
   (g) openid.signed: with the signed fields
   (h) openid.sig with a Base64 encoded signature

   The assertion can include also some optional parameters like the OP Local Identifier.

7. The RP verifies the information received including checking the return URL, verifying the discovered information, checking the nonce, and verifying the signature.
2.5 OAuth

OAuth is an open authorisation standard. Although it is not a SSO solution, it could be part of the whole solution and I considered necessary explain what is it and how it works to understand whether it fits or not in the solution.

OAuth was born in 2006 when developing the implementation of OpenID for Twitter, when the developers realised there was no open standards for APIs access delegation. It was in 2007 when some implementers purposed to write a draft of the implementation of OAuth. On October of the same year a final draft was released. But it was in 2010 when the OAuth standard was published as RFC and all the applications in Twitter required the use of OAuth. The current version of OAuth is the 2.0. This version is not compatible with the first one, and it is focused on the development simplicity to help developers integrate easily OAuth in their software. As new web software, desktop applications and mobile technologies are appearing constantly; OAuth 2.0 is focused on providing a simple way to implement it in the new software. Some of the most important companies in the world already use OAuth as an authorisation method, for example, Google, Facebook, Twitter, Foursquare, Instagram, Yahoo!, LinkedIn, MySpace, GitHub, Microsoft or Vimeo.

2.5.1 OAuth Protocol

To understand how OAuth works we have to redefine the different components that take part in the procedure. The user is the person that requests the information, as in OpenID. The Consumer is the application that tries to access to protected information. It is where the useful part of the action happens. Then, the SP is the one that has the restricted information and the one who the Consumer requests the information. Once defined the roles, we are going to describe the authorisation procedure. To explain it better, first there will be an overview of an OAuth situation of a requested protected information and after the explanation we are going to enter in more detail into the protocol.

2.5.1.1 OAuth Overview

We will use an example to make easy to understand the OAuth procedure before entering in the details. In this example, there is a user that has some photos and their description in a website. This website is the SP, is the one that has the protected resources. But the user wants to use the new web application to add some effects onto the photos.
This web application is the Consumer, the website that is going to request the protected resources. Assuming that the photos are already uploaded to the SP, we are going to explain how the user gets them.

1. The user wants to allow the Consumer use some resources stored at the SP’s server. For example, the user wants to use the photos stored at the SP and thus, the SP answers saying that the requested resource is protected, so it is not allowed to use the photos yet.

2. The Consumer redirects the user to the SP’s webpage. The user logs in if necessary. The login procedure depends on the SP. Remember that OAuth is about authorisation, not about authentication.

3. The SP asks the user to let the Consumer use the requested information. In this case, the requested information is the photos the Consumer wants to add the effects.
4. If the user agrees, the SP redirects the user to the Consumer’s webpage adding a token used to accesses the protected information.

5. The Consumer uses the token as defined in the OAuth specification and access to the information. Now, the Consumer is able use the photos requested from the SP to add the effects.

### 2.5.1.2 OAuth Specification Details

Although there are currently two versions of OAuth currently used, I’m going to explain the newest one. Basically both versions of the protocol are valid and widely used, but they are not compatible with each other. Although OAuth 1.0 is used at many websites, the second version of this protocol expands some existing features and adds new ones. There were also some points where OAuth 1.0 was a bit confusing and limited like in the authentication, the user experience and scalability. Also, the lack of a good documentation made some developers confused, especially, when working on the client side. That’s why OAuth 2.0 was developed and now some of the most known websites such as Facebook or Twitter use it as a standard.

Again to understand how OAuth works we have to define the different roles in the process as they are named in the draft.

- **RO**: entity that can provide access to a protected resource. When it is a person is referred as end-user.

- **RS**: server that hosts the protected resource. This server has to be able to provide the protected requests using access tokens.

- **Client**: application that makes the request on behalf of the RO and with its authorization.

- **Authorization server**: the server that provides access tokens to the client after authenticating the RO.

The authorization server and the RS may or may not be the same server. The same authorization server can provide access tokens for different RSs, though.

It is important to know that all the HTTPS requests are over SSL because it guarantees that the information shared, such token codes and secrets, is not compromised for other users to steal it. Going deeper in the OAuth implementation, the following steps will explain with more detail the authorization process.
Before start with the User case, both RO and Client need to share the compromise of being one part of another. To make this possible, the Client creates an OAuth Client in its server and requests the RO for the credentials, sending its app_name and a callback URL. The RO creates the necessary credentials in the RS, adding the Client as OAuth Client and generating a client_id and a client_secret. Then the RO sends the client_id and the client_secret using a secure way. Next, the Client adds the RO credentials to its system, so that they both share the same id and secret.

Once the RO and the Client are synchronized it is time to check how the process is done when a User requests some data. First, the User asks the Client for some data stored in the RS. The Client response is that it has no access yet to retrieve the data, so it responds with an authorize URL, a redirect URI, a client ID and the response type. The browser reads the response and pops up a token-not-found window that redirects to RS. The browser sends the client_id, the response type and the redirect URI. The RS check if the user is authenticated, and if it is not, opens a login window and an authorization message along with the previous information and adding authorize=1 if the User authorizes it.

After this, the RS generates an authorization code. This is not the token, it is just to know that the User authorizes the Client to request the access token to the RS. The RP responds with a redirect URL. It adds the authorization code and expiration into the URL. This URL is sent back to the Client through the Browser. Once the Client knows the authorization code and it has an authorization code also in the RS, it is time to request the token. The authorization code can be public, as it needs the client_id and client_secret to request the data. So, the Client sends a POST request to the RS with the client_id, the client_secret, the authorization code, the redirect URL and the grant type that says it is an authorization code. The RS creates an authorization token and revokes the authorization code. The RS returns then the access_token, a refresh_token and an expiration time in JSON format.

The Client has finally access to the data sending a HTTPS request with the token in the header and asking for the URL that retrieves the information. Finally, the Client manages the data and shows it to the User.
Chapter 3

Liferay

3.1 Introduction to Liferay

Many new web technologies have emerged to improve web portals performance. The CMS are one of these improvements. CMS provide good tools to develop web portals and easily customize them, easing the web developers work. Commonly CMS provide extensions and tools such as Really Simple Syndication (RSS) feeds, language translation and user administration among others. These CMS are developed in different programming languages such as PHP, Ruby, J2EE or Python. They give a basic structure of a web portal and let the developer expand it using existing extensions, customized themes.

However, Liferay is a Content Management Framework. CMFs include portlets. Portlets are truly useful and powerful for portal developers, as CMS can integrate a piece of full-customizable software inside their structure without modifying the rest of the portal, and being totally independent. These portlets can also be exported to other portals using the same CMF, and dont need to restart the server every time. The difference between other Frameworks and CMF is that the CMF provides a more defined UI and lets the developer focus on the portlet development instead of the whole look of the portal. Some disadvantages of the CMF in comparison with Frameworks are less flexibility in the UI and more difficulty in modifying extensions.

3.2 Liferay Portal Development

In Liferay, there are many different parts or functionalities that can be created and modified. These things can be an extension of some functionality that Liferay already implements, it can be a new small feature or the main part of our web application.
Liferay provides different ways to implement each one of those functionalities. There are also some tools to customize the look and feel of the webpage, such as themes or layout templates, but we are going to focus mostly in the practical web development more than the design and interface of the overall website.

### 3.2.1 Portlets

Portlets are small web applications that run in a certain space of a webpage. They are the most important part of a portal, because they contain the real functionality and the main applications. Liferay provides portlet containers that allow appearing portlets inside a particular page.

Portlets are the most independent and the less invasive form to run a piece of code in a webpage. They are also forward compatible with most of the old versions and they are hot-deployed as plugins, which means quick deployment and many facilities to the developer. A single plugin can contain more than one portlet in order to split the functionalities into various pieces that the webpage can load dynamically.

There are many technologies that portlets support. All technologies compatible with Java are supported; so you can use Struts, Spring, PHP or Ruby frameworks.

### 3.2.2 Ext plugins

Ext plugins provide the most flexible way to modify the Liferay core. This kind of plugin lets the developer modify any kind of class with its custom implementation. On the other hand, Ext plugins are mostly written for a concrete version of Liferay. This is why this plugins are only recommended for really necessary modifications that the developer cannot do in any other way. If the Liferay version needs to be upgraded, probably the Ext plugin will need major modifications in order to work properly. This is why upgrading Ext plugins is not recommended. In earlier versions of Liferay, Ext plugins did not exist and Extension Environments were used instead. After Liferay 6.0, Ext plugins were introduced as real plugins. Another reason why Ext plugins have to be used only in indispensable cases is that the deployment of these plugins needs the server to restart to take effect.
Figure 3.1: Portlet screenshot
Chapter 4

Solution analysis

4.1 Requirements of the solution

There are basically few things to take in count when we want to define the needs of this login system. Going back to the Q1 defined at the beginning of this Thesis, we need to know the main features that the login system must have, in order to choose the best login system that fits into our portal. These properties are going to be defined as we describe the needs.

It is likely that new partners are going to join the system, so the system has to be prepared for that. So the requirement R1: **it has to be scalable.** This means that at any time a new partner can be added to the system without major modifications, or without any modification. It has to be easy to make it bigger and it has to be prepared even if it has only one partner or it has one thousand of them.

As the system has to be scalable, it is likely that there will be many partners. These partners are going to implement the same login method than the portal. It is for this reason that the selected SSO method must accomplish the second requirement R2: **is has to be easy to implement.** As all the partners will have to implement the same method, there are more possibilities that they accept to implement it if the effort in the implementation is less. Moreover, most of the partners will use some CMS or Framework to develop their website, so the selected method should have as much libraries as possible in order to provide them high flexibility and high freedom degree when they are developing the website.

Most of the information provided is confidential and has to be protected. Thus, the login system is a critical part if we want to protect the data from attacks. The selected method’s requirement R3 is: **it has to be totally secure and use encrypted**
data to manage the login system. This is really important and as the information handled involves private information about the users, the system has to prevent attacks and provide a secure way to authenticate the user without compromising the privacy of the data sent.

There are also another minor requirements that should be taken in count when selecting the login method. There is not a concrete lifetime for the portal, and thus the login system has to be available as long as the portal exists or even further. To enable this, the SSO system has to be a lifelong system managed by the owners of the portal. This means that it cannot be managed by someone else, another institution, organisation, company, etc. This kind of system is called decentralized, because there is not a central institution, organization or company that manages it. So the requirement **R4: the selected SSO must be decentralized.** Even most of the centralized systems are important institutions and organizations, the reliability of their system is relative, and it can never be less independent than if the system used is decentralized. Furthermore, the login information will be handled by a third party, which is not very pleasant for the portal owners as we have to trust in someone else's systems. But anyway, the decentralization of the system is always a plus and even it has to be one of the points in the decision it is not a major issue.

### 4.2 Analysis

Once described the main SSO solutions purposed, it is time to decide which one of them is the best solution, answering the Q1. All the Single Sing-on methods described above are going to be analysed and compared. The main thing that we should consider is that we need a login system that fits within our needs. Considering a wrong login system could head the development of the project to an unnecessary and complex process. Again all the SSO systems are going to be mentioned, but this time they are also going to be discussed and compared with each other.

#### 4.2.1 Social Security Number

This was one of the first solutions discussed, but we quickly realised that there are many problems with it. The first and the most important is the security hole of this system. The Social Security number can be intercepted by someone and used to forge the identity of the user. With an easy attack, someone can know the Social Security number of a user. Using this number the attacker can login into the partners website in the same
way that the Web Portal would do it. The lack of security is, evidently, the main reason why this option was discarded.

There are also other minor issues. For example, the Social Security number cannot be a good unique identifier. It might happen that for some reason a person doesn't have a Finnish Social Security number. As the idea is to make it scalable, it also can happen that more than one person has the same Social Security number in different countries. It also can happen that one person has no Social Security number as long as in Finland is not required for short-term stays.

4.2.2 Shibboleth

Shibboleth seems to fit quite well to solve our problem. It provides the same authentication method for different websites, creating sessions that can be used in more than one page. It also solves the security problem mentioned with the Social Security number solution and provides a reliable authentication, proved by educational and governmental institutions among others. It also solves other problems like having a unique identifier for each user and it makes the Web Portal scalable if the partners websites can provide a Shibboleth authentication system in their websites.

However, there are some aspects to take in count. Shibboleth is a centralized system, which means that the IP is an organisation that provides the user authentication. That means that the lifetime of the authentication method depends on the lifetime of this organisation, and as long as it stops existing the authentication method won't work. It is a minor issue, but can still be improved.

4.2.3 OpenID

According to the needs of the Web Portal, OpenID seems to fit really well in the requirements to use a SSO system. Using OpenID is a widely extended way to solve the SSO problem. For that reason, most of the common CMS and most of the web technologies implement OpenID libraries to make it easier for developers. That is why it is way easier to sell to the partners to integrate it in their websites. In the Liferay CMS there is already implemented an OpenID library that makes the integration easier.

This solution also fulfill the other requirements. As in Shibboleth, OpenID provides a reliable and secure authentication method. Each user that already exists can be linked to an OpenID account just by adding a new field to the database, solving the problem of having several fields for each partners websites login. Moreover, the users already
created are not going to lose all their user information in the database. This makes the solution scalable and easy to integrate both new users and new partners.

The good point of OpenID is the decentralisation. That means that the OpenID Provider does not depend on any third-party organisation. An OpenID Provider can be implemented in an owned server. This makes it a non-dependent solution for the login and all the partners websites can integrate an authentication system settled in a known server.

![Figure 4.1: Schema of OpenID](image)

### 4.2.4 OAuth

When connecting different web portals to each other we have to take in count the way they are going to be connected, and how are you going to use the data provided by the other websites. In our case, we use iFrames to display the data provided by the other websites, so we just display the partners website without any kind of hidden data transfer from the partners website to our portal. All the necessary information to display the website is provided within the HTTP request via GET or POST.

OAuth let you access to some information provided by the Resource Owner. If we wanted to use OAuth, the portal should provide an API to access to its resources and its going to become the Resource Owner. Then, our partners must use an authentication system to access to these resources. In this case, every partner should authenticate at least the first time to access to the API. Then the partner’s portal, that become the Clients, request the Resource Owner who is using the application and some other necessary data. When the Client receives this information, they use it to display the appropriate webpage. For
example, if nurse that wanted to see some information about a patient, the partner’s portal would have authenticate and allow this partner to use her information if it is the first time, then retrieve the information about who is the nurse and who is the patient, and finally render the information.

As we can see, this mechanism is way much difficult to implement. All partners have to implement all this mechanism to retrieve the data and display the appropriate webpage. Usually all these methods are useful for big amounts of data, but in the case of this portal it is not necessary. Since the information exchanged is small, all those steps are tough and there is a much easier way to display this just staying logged in and asking for a concrete URL that contains these parameters.

### 4.3 Comparison of the SSO Systems

After having analysed the purposed SSO systems, we can compare them to find out which one fits better within our needs. In order to display this comparison clearly, the table below shows a brief overview of the pros and cons of each system. The Social Security Number method can be avoided and marked as clearly unsafe. Henceforth this method is not going to be used nor studied as a real option in order to focus in the priorities of answering the Q1.

<table>
<thead>
<tr>
<th></th>
<th>Shibboleth</th>
<th>OpenID</th>
<th>OAuth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scalability</strong></td>
<td>✔ New partners can be added without extra effort.</td>
<td>✔ New partners can be added without extra effort.</td>
<td>✔ New partners can be added with relative ease.</td>
</tr>
<tr>
<td><strong>Simplicity</strong></td>
<td>✔ As SAML is a standard, it is widely supported for web development.</td>
<td>✔ OpenID is an open standard widely supported in most of the programming languages.</td>
<td>✗ OAuth is an open standard. However, it can be tricky to implement. It is less trivial for partners.</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>✔ It is considered as secure and some reputed companies use it.</td>
<td>✔ It is considered as secure as no bugs have been found in a long time and big companies use it.</td>
<td>✔ It is considered as secure as no bugs have been found in a long time and big companies use it.</td>
</tr>
<tr>
<td><strong>Independency</strong></td>
<td>✗ It depends on a global organization.</td>
<td>✔ Totally independent because of the use of OpenID Providers.</td>
<td>✔ Totally independent. User info is save within the same website.</td>
</tr>
</tbody>
</table>

**Figure 4.2:** Comparative table of the different SSO

As seen in the table above, in scalability matters, all the solutions provide a scalable solution. All the scalable methods have many things in common. First, there is a common IP that authenticates the user and gives the credentials to the portal. This
is a vital point because every user of these systems does not have to take care of the authentication process because it is all done in one IP. They only have to manage the response of the IP whether it is correct or not.

The simplicity of implementation varies. For example Shibboleth uses OpenSAML, which can be used easily in C++ and Java. OpenID is also widely supported in most of the languages. There many libraries for the most important programming languages such as Java, C++, Python, Ruby, PHP, C#, ColdFusion, Perl and others. OAuth has also lots of libraries in different languages like OpenID, further adding some in Javascript and Objective-C among others. However, as some experts recognize, OAuth can be a bit tricky especially in the earlier versions. Moreover, as OAuth supports only authorization, the implementation is more tedious because it has to implement an authentication method anyway and then also the information exchange that should be managed by the partner, displaying whatever is request.

One thing everybody knows is that an authentication system has to be secure. All the information stored is confidential, and having an unsafe authentication method could lead the project to the failure and some legal problems. This is why all the login systems compared above are considered as safe standards. Even this point can be obvious, it needs a special mention to clarify it.

The last point talks about the independency of the system regarding another third parties that can manage the login system. As shown, Shibboleth is the only one that has a central organization that provides the authentications. As discussed before, this can be a weak point because we are relying in someone else’s security and most of the partners may not like it. Moreover, the availability of the login system is clearly uncertain, as there is not a true sureness that the service will be available forever or at least the time we are going to need this system to be available. So, Shibboleth has a clear disadvantage compared to OpenID and OAuth.

### 4.4 Selected SSO System

After a deep comparison between the most suitable login systems and their descriptions and performance, the conclusion is that the best solution for the needs of the portal is the integration of OpenID as a login system. The comparative table in the previous section shows clearly that OpenID is the login system that fulfils the requirements of our needs. Moreover, its implementation fits perfectly in our Liferay portal due to its inclusion of an OpenID library and a good integration with the CMF.
There are many reasons why OpenID fits much better than the other solutions. According to the analysis made, we can observe that OpenID has similar points with Shibboleth in its procedure. Taking apart obvious differences, the main one is the vastly mentioned decentralization. This is a key point when deciding between them. As there is no need to retrieve information from the portal that cannot be sent by GET or POST, there is no need to provide an API to retrieve this information and consequently, there is no need of using OAuth.

Another advantage of OpenID is the Liferay integration. Liferay provides OpenID libraries and a complete integration in the UI. So the integration with the portal and the partners portals that are made with Liferay will easily integrate this solution without major efforts.
Chapter 5

Implementation

5.1 How to implement OpenID in Liferay

Once the SSO is known we can focus on the second question, the Q2, and talk about the implementation of OpenID in Liferay. Once explained what is Liferay and how does it work, we can proceed to explain the different tools that Liferay provide and how they are used to implement OpenID.

Liferay has changed a lot along its new versions. Old implementations are way trickier and did not have many helps. However, this has been improved along new versions. In the Liferay version that the portal is done, the 5.2.3, there are still some old tools that nowadays have been improved. There are mainly two sides that have been implemented: the Client side and the Provider.

5.2 The Client

In this case, the Clients are all the websites integrated that are going to use the same OpenID login system. So that includes both the Liferay portal, which is going to be explained, and the partners’ websites.

Focusing on the Liferay implementation and how the current portal is done, there are three main parts. The first one is the core of the portal. It is developed using an External Environment of Liferay. In the developed Ext, there are all the data models and functionalities of the portal, including the different type of users and their roles, permissions and related information. This Ext is where all the data is managed to be displayed depending on the user, but is also where the login methods take place. So is here where some modifications are needed in order to adapt the login system to OpenID.
The second part of the implementation is the look of the page, done with JSP. Here is where the data managed by the core is displayed. Using JSP, CSS and HTML, the data is taken from the core of the portal and rendered to the templates.

The third part is the database. This Liferay portal is implemented using MySQL as a RDBMS. We have to make some manual adjustments for the existing users to link them their OpenID account. This procedure will not be necessary for new users, as Liferay provide an OpenID field for including the OpenID Identifier when registering new users.

### 5.2.1 Using OpenID Java Library

This portal should be used also for elderly people and people who don’t know much more about computers, so the login system has to be as simple as possible. To keep it easy, the login should be just a username, instead of all the OpenID Identifier, that is a URL. To manage this, this piece of code has to be included before sending the request to the OP. The only thing needed is to take the value of the openId field and add the URL of the OP.

Moreover, if we want to access another OP, for administration purposes for example, writing the whole OpenID Identifier will be enough so that not only our own OP is the only one that can be accessed. However, even if we try to login to another OP, if the OpenID Identifier is not linked into any user, then it won’t have any practical effect.

This is the piece of code:

```java
// OpenIdAction.java
// Obtain the openId String
String openId = ParamUtil.getString(actionRequest, "openId");

// Adapt the query to the default OP
if (!openId.startsWith("http://")) openId = "http://localhost:8080/openid/" + openId;
```

Once the request is sent, the OP is going to send the response. The Java code will handle the response. By default, as we call the openId method, the response redirect us to the OpenID default login page of Liferay. To change that, the following lines must be modified:

```java
try {
    if (cmd.equals(Constants.READ)) {
        String redirect = readOpenIdResponse(
            themeDisplay, actionRequest, actionResponse);
```
if (Validator.isNull(redirect)) {
    redirect =
        PortalUtil.getPortalURL(actionRequest) +
        themeDisplay.getURLSignIn();
}

sendRedirect(actionRequest, actionResponse, redirect);
} else {
    sendOpenIdRequest(themeDisplay, actionRequest, actionResponse);
}
}

catch (Exception e) {
    if (e instanceof DuplicateUserEmailAddressException) {
        SessionErrors.add(actionRequest, e.getClass().getName());
    } else if (e instanceof OpenIDException) {
        if (_log.isInfoEnabled()) {
            _log.info("Error communicating with OP: " +
                        e.getMessage());
        }
        System.out.println(e.getClass().getName());
        SessionErrors.add(actionRequest, e.getClass().getName());
        String redirect = readOpenIdResponse(
            themeDisplay, actionRequest, actionResponse);

        if (Validator.isNull(redirect)) {
            redirect =
                PortalUtil.getPortalURL(actionRequest) +
                themeDisplay.getURLSignIn();
        }
        sendRedirect(actionRequest, actionResponse, redirect);
    } else {
        _log.error("Error processing the OpenID login", e);
        PortalUtil.sendError(e, actionRequest, actionResponse);
    }
}

As we can see, also if there is some error handling the response of the OP, the code will redirect the user to the customized login page and not to the default OpenID login page.
There is also one issue pending. If the OpenID Identifier returned by the OP is not into the system, the new user can be introduced to the system or can the request can be denied. This point can be modified depending on the preferences and policies of the portal. Here is where it can be modified, but it is currently implemented as the user can be registered.

```java
try {
    user = UserLocalServiceUtil.getUserByOpenId(openId);
}
catch (NoSuchUserException nsue) {
    if (Validator.isNull(firstName) || Validator.isNull(lastName) ||
        Validator.isNull(emailAddress)) {
        SessionMessages.add(request, "missingOpenIdUserInformation");

        if (_log.isInfoEnabled()) {
            _log.info("The OP did not send the required " +
                "attributes to create an account");
        }

        PortletURL createAccountURL =
            themeDisplay.getURLCreateAccount();

        createAccountURL.setParameter("openId", openId);

        session.setAttribute(
            WebKeys.OPEN_ID_LOGIN_PENDING, Boolean.TRUE);

        return createAccountURL.toString();
    }
    long creatorUserId = 0;
    long companyId = themeDisplay.getCompanyId();
    boolean autoPassword = false;
    String password1 = PwdGenerator.getPassword();
    String password2 = password1;
    boolean autoScreenName = true;
    String screenName = StringPool.BLANK;
    Locale locale = themeDisplay.getLocale();
    String middleName = StringPool.BLANK;
    int prefixId = 0;
    int suffixId = 0;
    boolean male = true;
```
int birthdayMonth = Calendar.JANUARY;
int birthdayDay = 1;
int birthdayYear = 1970;
String jobTitle = StringPool.BLANK;
long[] groupIds = null;
long[] organizationIds = null;
long[] roleIds = null;
long[] userGroupIds = null;
boolean sendEmail = false;

ServiceContext serviceContext = new ServiceContext();

user = UserLocalServiceUtil.addUser(
    creatorUserId, companyId, autoPassword, password1, password2,
    autoScreenName, screenName, emailAddress, openId, locale,
    firstName, middleName, lastName, prefixId, suffixId, male,
    birthdayMonth, birthdayDay, birthdayYear, jobTitle, groupIds,
    organizationIds, roleIds, userGroupIds, sendEmail,
    serviceContext);

Once modified the core implementation, let’s have a look to the actual view of the portal.

5.2.2 Forms and JSP

The External Environment provided with the Liferay portal already provides a customization of the login page. With some modifications to the current login page it is possible to call OpenID methods. The main change will be the login form. The old form included one username field and one password field. The new one includes just one field with the OpenID username, not even the entire OpenID Identifier, to ease the user’s work and improve the user experience.

First there are some variables that are necessary for the form. These ones:

```jsp
<% 
    String redirect = ParamUtil.getString(renderRequest, "redirect");
    String openId = ParamUtil.getString(request, "openId");

    String login = LoginUtil.getLogin(request, "openId", company);
    boolean rememberMe = ParamUtil.getBoolean(request, "rememberMe");

    if (Validator.isNull(authType)) {
```
authType = company.getAuthType();
}

Once the variables are ready, the next step is calling the Liferay Framework to request the URL where the form has to be sent. The struts_action is the method is going to be called to proceed with the login.

<portlet:actionURL var="openIdURL">
    <portlet:param name="saveLastPath" value="0" />
    <portlet:param name="struts_action" value="/login/open_id" />
</portlet:actionURL>

Now, the variable openIdURL contains the URL where the form is going to be sent. After that, the OpenID form has to be rendered using HTML and filling the fields with the retrieved variables.

<form action="<%= openIdURL %>">
    <div class="form-row">
        <div class="label"><label for="openId">Tunnus</label></div>
        <div class="openid-login"><input name="openId" type="text" value="<%=HtmlUtil.escape(login)%>" id="openId" />
        </div>
    </div>
    <div class="clear"></div>
    <div class="submit-button"><input type="submit" value="<liferay-ui:message key="sign-in" />
        </div>
    <input name="redirect" type="hidden" value="<%= redirect %>">
</form>
As we can see, all the required OpenID components are in the form, including the OpenID method where the form is sent, the OpenID Identifier (in this case, the OpenID username), the submit button and the redirect page.

Now the login page renders a field that has to be filled with a username, and sends it correctly to the OP.

### 5.2.3 The Database

The last part of the implementation is about the Database. At this point the portal is ready for new users who have an OpenID account linked to their user, but for the existing users there is no such link. So after knowing the OpenID Identifier and once opened an account for the existing user, the database must be modified to link the existing user with its OpenID Identifier.

As the RDBMS used is MySQL, some tool to manage it can be downloaded or directly from the console. Inside the lportal database, there is a table named user_. The only thing is to modify the openId field filled with the OpenID Identifier in the form of
URL. Doing this the selected user will be linked to its OpenID account and logged in successfully as a user of the portal if the OP accepts the authentication.

![The Users table and the OpenID field](image)

**Figure 5.2:** The Users table and the OpenID field

### 5.3 The OpenID Provider

This is a really important part of the OpenID integration with the portal. As other partners will rely on this OP as an authentication system, it has to be secure, consistent and unbreakable. In order to accomplish this, there are lots of open-source OPs in different languages widely tested. The tests done by the Providers are way more exhaustive and secure because they are implemented and tested by experts, so there is no need to code an OP from the beginning.

After analysing many Java OPs, the most consistent and supported one was JOIDS. JOIDS is based in OpenID4Java, Spring Framework, Hibernate and Velocity. It implements two ways to display your OpenID Identifier, one as a folder of the domain and other as a subdomain. It also includes account management and multi-user and multi-domain support.

To install JOIDS, it has to be included into the deploy folder of Apache Tomcat. Once Tomcat has started, it will deploy the WAR file and start it. Then, inside the JOIDS
folder there is a file to configure JOIDS. In order to use MySQL and setup a password to activate JOIDS the following lines must be modified:

```xml
<Resource name="jdbc/AlhPool" auth="Container" type="javax.sql.DataSource"
  driverClassName="com.mysql.jdbc.Driver"
  url="jdbc:mysql://localhost/alh?useUnicode=true&characterEncoding=UTF-8"
  username="root" password="xavi" maxActive="20" />

<Environment
  name="domain.configurator.password"
  type="java.lang.String"
  value="activelife"
  override="false" />
```

Once changed, the browser will display a page that lets the user configure the domain. It is now when the previous password needs to be introduced. After introduced the password, just few adjustments are left. First, select the type of URL that will belong to each user. Then, it is done. The OP is configured to manage the users. If the registration feature needs to be banned to make sure that no one can register, change the necessary permissions to access the file.
Chapter 6

Conclusions

Technology is growing by giant steps. However, there are people that show themselves aversive to technology. There is a statistical relation between the age of these people and their difficulties in learning new technology. The older a person is, the more aversive is to abstract technological concepts. Anyway, this problem will be probably solved for the next generations, which have more facility managing new devices and understanding new technological concepts. As this project is oriented to improve elderly people’s life, the user interaction has to be as easy as possible. This point is really important if we want to build a useful portal that actually eases the work instead of hinder it.

Research question Q1 handled different solutions: which is the most suitable SSO solution for the Web Portal? One of the vital parts of the portal is the integration of the external services and the need of a unique way to login the users. In this way, OpenID fulfilled all the requested features needed to build a good login system. So in response to the Q1, the selected option has demonstrated to be the best solution on paper and in the real case.

Research question Q2 concerned about the implementation: how to implement the selected SSO solution? Over other options, OpenID has the advantage of being easy to implement, maybe the easiest one. The importance of this is related to the number of partners in some way. The easier it is to implement the login system, the more likely it is to be adapted to the partner’s website. Some companies can see a tricky implementation as a big challenge and a trouble, so it can be seen as an obstacle that makes them decide whether to attach their portal as a partner or not. This is why answering the Q2 was also important. Through the Q2 we can see that the implementation of OpenID in Liferay is quite easy. The difficult thing is get used to develop Liferay portals because it requires a lot of time to clearly understand how everything works. Although the documentation is very poor and the forum threads are not even answered in most of the
cases, after observing the implementation of some examples and the way the External Environment is coded we can realize how Liferay works. But the difficulty lies more in the Liferay development than in the OpenID libraries, which is good for partners that may have their web developers used to their programming environment and the only new thing they have to learn is to use the OpenID libraries.
Appendix A

OpenID Core Code

/**
 * Copyright (c) 2000-2009 Liferay, Inc. All rights reserved.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to
 * deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE
 * SOFTWARE.
 */

package com.liferay.portlet.login.action;

import java.util.Calendar;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Set;
import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.PortletConfig;
import javax.portlet.PortletURL;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.compass.core.util.backport.java.util.Arrays;
import org.openid4java.OpenIDException;
import org.openid4java.consumer.ConsumerManager;
import org.openid4java.consumer.VerificationResult;
import org.openid4java.discovery.DiscoveryInformation;
import org.openid4java.discovery.Identifier;
import org.openid4java.message.AuthRequest;
import org.openid4java.message.AuthSuccess;
import org.openid4java.message.MessageExtension;
import org.openid4java.message.ParameterList;
import org.openid4java.message.ax.AxMessage;
import org.openid4java.message.ax.FetchRequest;
import org.openid4java.message.ax.FetchResponse;
import org.openid4java.message.sreg.SRegMessage;
import org.openid4java.message.sreg.SRegResponse;
import com.liferay.portal.DuplicateUserEmailAddressException;
import com.liferay.portal.NoSuchUserException;
import com.liferay.portal.kernel.language.LanguageUtil;
import com.liferay.portal.kernel.log.Log;
import com.liferay.portal.kernel.log.LogFactoryUtil;
import com.liferay.portal.kernel.servlet.SessionErrors;
import com.liferay.portal.kernel.servlet.SessionMessages;
import com.liferay.portal.kernel.util.Constants;
import com.liferay.portal.kernel.util.GetterUtil;
import com.liferay.portal.kernel.util.ParamUtil;
import com.liferay.portal.kernel.util.StringPool;
import com.liferay.portal.kernel.util.Validator;
import com.liferay.portal.model.User;
import com.liferay.portal.service.ServiceContext;
import com.liferay.portal.service.UserLocalServiceUtil;
import com.liferay.portal.struts.PortletAction;
import com.liferay.portal.theme.ThemeDisplay;
import com.liferay.portal.util.OpenIdUtil;
import com.liferay.portal.util.PortalUtil;
import com.liferay.portal.util.WebKeys;
import com.liferay.portlet.ActionResponseImpl;
import com.liferay.util.PwdGenerator;

/**
 * <a href="OpenIdAction.java.html"><b><i>View Source</i></b></a>
 * *
 * @author Brian Wing Shun Chan
 * @author Jorge Ferrer
 * *
 */
public class OpenIdAction extends PortletAction {

    public void processAction(
        ActionMapping mapping, ActionForm form, PortletConfig portletConfig,
        ActionRequest actionRequest, ActionResponse actionResponse)
            throws Exception {

            ThemeDisplay themeDisplay =
            (ThemeDisplay)actionRequest.getAttribute(
                WebKeys.THEME_DISPLAY);

            if (actionRequest.getRemoteUser() != null) {

                actionResponse.sendRedirect(themeDisplay.getPathMain());

                return;
            }

            String cmd = ParamUtil.getString(actionRequest, Constants.CMD);

            try {
                if (cmd.equals(Constants.READ)) {

                }
            }

}
String redirect = readOpenIdResponse(
    themeDisplay, actionRequest,
    actionResponse);

    if (Validator.isNull(redirect)) {
        redirect =
        PortalUtil.getPortalURL(actionRequest) +
        themeDisplay.getURLSignIn();
    }

    sendRedirect(actionRequest, actionResponse,
        redirect);
  } else {
    sendOpenIdRequest(themeDisplay, actionRequest,
        actionResponse);
  }
}

catch (Exception e) {
    if (e instanceof DuplicateUserEmailAddressException) {
        SessionErrors.add(actionRequest,
            e.getClass().getName());
    } else if (e instanceof OpenIDException) {
        if (_log.isInfoEnabled()) {
            _log.info(
                "Error communicating with
              OpenID provider: " +
                e.getMessage());
        }
        System.out.println(e.getClass().getName());
        SessionErrors.add(actionRequest,
            e.getClass().getName());
        String redirect = readOpenIdResponse(
            themeDisplay, actionRequest,
            actionResponse);

        if (Validator.isNull(redirect)) {
            redirect =
            PortalUtil.getPortalURL(actionRequest) +
        } else {
            sendOpenIdRequest(themeDisplay, actionRequest,
                actionResponse);
        }
    }}
themeDisplay.getURLSignIn();
    }
    sendRedirect(actionRequest, actionResponse, redirect);
    } else {
        _log.error("Error processing the OpenID login", e);
        PortalUtil.sendError(e, actionRequest, actionResponse);
    }
}

public ActionForward render(
    ActionMapping mapping, ActionForm form, PortletConfig portletConfig,
    RenderRequest renderRequest, RenderResponse renderResponse)
    throws Exception {

    ThemeDisplay themeDisplay =
        (ThemeDisplay)renderRequest.getAttribute(
            WebKeys.THEME_DISPLAY);

    renderResponse.setTitle(
        LanguageUtil.get(
            themeDisplay.getCompanyId(),
            themeDisplay.getLocale(),
            "open-id"));

    return mapping.findForward("portlet.login.open_id");
}

protected String getFirstValue(List<String> values) {
    if ((values == null) || (values.size() < 1)) {
        return null;
    }

    return values.get(0);
}
protected boolean isCheckMethodOnProcessAction() {
    return _CHECK_METHOD_ON_PROCESS_ACTION;
}

protected String readOpenIdResponse(
    ThemeDisplay themeDisplay, ActionRequest actionRequest,
    ActionResponse actionResponse)
throws Exception {

    HttpServletRequest request = PortalUtil.getHttpServletRequest(
        actionRequest);
    HttpSession session = request.getSession();

    ActionResponseImpl actionResponseImpl =
        (ActionResponseImpl)actionResponse;

    ConsumerManager manager = OpenIdUtil.getConsumerManager();

    ParameterList params = new ParameterList(
        actionRequest.getParameterMap());

    DiscoveryInformation discovered =
        (DiscoveryInformation)session.getAttribute(WebKeys.OPEN_ID_DISCO);

    if (discovered == null) {
        return null;
    }

    PortletURL portletURL = actionResponseImpl.createActionURL();

    portletURL.setParameter("struts_action", "/login/open_id");
    portletURL.setParameter(Constants.CMD, Constants.READ);
    portletURL.setParameter("saveLastPath", "0");

    VerificationResult verification = manager.verify(
        portletURL.toString(), params, discovered);

    Identifier verified = verification.getVerifiedId();

    if (verified == null) {
        return null;
    }
AuthSuccess authSuccess = (AuthSuccess)verification.getAuthResponse();

String firstName = null;
String lastName = null;
String emailAddress = null;

if (authSuccess.hasExtension(SRegMessage.OPENID_NS_SREG)) {
    MessageExtension ext = authSuccess.getExtension(SRegMessage.OPENID_NS_SREG);

    if (ext instanceof SRegResponse) {
        SRegResponse sregResp = (SRegResponse)ext;

        String fullName = GetterUtil.getString(sregResp.getAttributeValue("fullname"));

        int pos = fullName.indexOf(StringPool.SPACE);
        if ((pos != -1) && ((pos + 1) < fullName.length())) {
            firstName = fullName.substring(0, pos);
            lastName = fullName.substring(pos + 1);
        }
    }
}

emailAddress = sregResp.getAttributeValue("email");

if (authSuccess.hasExtension(AxMessage.OPENID_NS_AX)) {
    MessageExtension ext = authSuccess.getExtension(AxMessage.OPENID_NS_AX);

    if (ext instanceof FetchResponse) {
        FetchResponse fetchResp = (FetchResponse)ext;

        if (Validator.isNull(firstName)) {
            firstName = getFirstValue(fetchResp.getAttributeValues("firstName"));
        }
    }
}
if (Validator.isNull(lastName)) {
    lastName = getFirstValue(fetchResp.getAttributeValues("lastName"));
}

if (Validator.isNull(emailAddress)) {
    emailAddress = getFirstValue(fetchResp.getAttributeValues("email"));
}

String openId = OpenIdUtil.normalize(authSuccess.getIdentity());

User user = null;
	ry {
    user = UserLocalServiceUtil.getUserByOpenId(openId);
} catch (NoSuchUserException nsue) {
    if (Validator.isNull(firstName) ||
        Validator.isNull(lastName) ||
        Validator.isNull(emailAddress)) {
        SessionMessages.add(request, "missingOpenIdUserInformation");

        if (_log.isInfoEnabled()) {
            _log.info("The OpenID provider did not send the required " +
                        "attributes to create an account");
        }

        PortletURL createAccountURL =
            themeDisplay.getURLCreateAccount();

        createAccountURL.setParameter("openId", openId);
    }
}
session.setAttribute(
    WebKeys.OPEN_ID_LOGIN_PENDING,
    Boolean.TRUE);

    return createAccountURL.toString();
}

long creatorUserId = 0;
long companyId = themeDisplay.getCompanyId();
boolean autoPassword = false;
String password1 = PwdGenerator.getPassword();
String password2 = password1;
boolean autoScreenName = true;
String screenName = StringPool.BLANK;
Locale locale = themeDisplay.getLocale();
String middleName = StringPool.BLANK;
int prefixId = 0;
int suffixId = 0;
boolean male = true;
int birthdayMonth = Calendar.JANUARY;
int birthdayDay = 1;
int birthdayYear = 1970;
String jobTitle = StringPool.BLANK;
long[] groupIds = null;
long[] organizationIds = null;
long[] roleIds = null;
long[] userGroupIds = null;
boolean sendEmail = false;

ServiceContext serviceContext = new ServiceContext();

user = UserLocalServiceUtil.addUser(
    creatorUserId, companyId, autoPassword,
    password1, password2,
    autoScreenName, screenName, emailAddress,
    openId, locale,
    middleName, male,
    jobTitle, groupIds,
    suffixId, serviceContext);
session.setAttribute(WebKeys.OPEN_ID_LOGIN, new Long(user.getUserId()));
return null;
}

protected void sendOpenIdRequest(
    ThemeDisplay themeDisplay, ActionRequest actionRequest,
    ActionResponse actionResponse)
throws Exception {

    if (!OpenIdUtil.isEnabled(themeDisplay.getCompanyId())) {
        return;
    }

    HttpServletRequest request = PortalUtil.getHttpServletRequest(
        actionRequest);
    HttpServletResponse response = PortalUtil.getHttpServletResponse(
        actionResponse);
    HttpSession session = request.getSession();

    ActionResponseImpl actionResponseImpl =
        (ActionResponseImpl)actionResponse;
    String openId = ParamUtil.getString(actionRequest, "openId");

    if (!openId.startsWith("http://")) openId = "http://" + openId + ".myopenid.com";

    PortletURL portletURL = actionResponseImpl.createActionURL();
    portletURL.setParameter("struts_action", "/login/open_id");
    portletURL.setParameter(Constants.CMD, Constants.READ);
    portletURL.setParameter("saveLastPath", "0");

    ConsumerManager manager = OpenIdUtil.getConsumerManager();

    List<DiscoveryInformation> discoveries = manager.discover(openId);

    DiscoveryInformation discovered = manager.associate(discoveries);
session.setAttribute(WebKeys.OPEN_ID_DISCO, discovered);

AuthRequest authRequest = manager.authenticate(
    discovered, portletURL.toString(),
    themeDisplay.getPortalURL());

try {
    UserLocalServiceUtil.getUserByOpenId(openId);
} catch (NoSuchUserException nsue) {
    String screenName = OpenIdUtil.getScreenName(openId);

    try {
        User user =
        UserLocalServiceUtil.getUserByScreenName(
            themeDisplay.getCompanyId(),
            screenName);

        UserLocalServiceUtil.updateOpenId(user.getUserId(), openId);
    } catch (NoSuchUserException nsue2) {
       FetchRequest fetch =
        FetchRequest.createFetchRequest();

        fetch.addAttribute(
            "email",
            "http://schema.openid.net/contact/email", true);
        fetch.addAttribute(
            "firstName",
            "http://schema.openid.net/namePerson/first",
            true);
        fetch.addAttribute(
            "lastName",
            "http://schema.openid.net/namePerson/last",
            true);

        authRequest.addExtension(fetch);

        SRegRequest sregRequest =
        SRegRequest.createFetchRequest();

        sregRequest.addAttribute("fullname", true);
sregRequest.addAttribute("email", true);

authRequest.addExtension(sregRequest);
}
}

response.sendRedirect(authRequest.getDestinationUrl(true));

private static final boolean _CHECK_METHOD_ON_PROCESS_ACTION = false;

private static Log _log = LogFactoryUtil.getLog(OpenIdAction.class);

}
Appendix B

JSP Login Form Code

<%  
/**  
* Copyright (c) 2000-2009 Liferay, Inc. All rights reserved.  
*  
* Permission is hereby granted, free of charge, to any person  
obtaining a copy  
* of this software and associated documentation files (the  
"Software"), to deal  
* in the Software without restriction, including without limitation  
the rights  
* to use, copy, modify, merge, publish, distribute, sublicense,  
and/or sell  
* furnished to do so, subject to the following conditions:  
*  
* The above copyright notice and this permission notice shall be  
included in  
* all copies or substantial portions of the Software.  
*  
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
EXPRESS OR  
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
MERCHANTABILITY,  
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT  
SHALL THE  
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR  
OTHER  
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,  
ARISING FROM,
 Appendix B. JSP Login Form Code

```jsp
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE * SOFTWARE.
*/
%

<%@ include file="/html/portlet/login/init.jsp"%>

<div class="css3rounded" style="width: 70%; margin-left: auto; margin-right: auto;">

```jsp
String redirect = ParamUtil.getString(renderRequest, "redirect");
String openId = ParamUtil.getString(request, "openId");

String login = LoginUtil.getLogin(request, "openId", company);
String password = StringPool.BLANK;
boolean rememberMe = ParamUtil.getBoolean(request, "rememberMe");

if (Validator.isNull(authType)) {
    authType = company.getAuthType();
}
```

```jsp
<h2>Kirjaudu sisn </h2>
</jsp>

<form action=" <%= openIdURL %>"class="standard-form" method="post" name="<portlet:namespace />fm" id="openid-form">
  <div class="form-row">
    <div class="label"><label for="<portlet:namespace />openId">Tunnus</label></div>
    <div class="openid-login"><input name="<portlet:namespace />openId" type="text" value="<%=HtmlUtil.escape(login)%>" id="<portlet:namespace />openId"/>
```
```
Appendix B. *JSP Login Form Code*

```html
</div>
</div>

<div class="clear"></div>

</div>

</div>

<div class="submit-button"><input type="submit" value="<liferay-ui:message key="sign-in" />
 id="submit-openid" /></div>

<input name="<portlet:namespace \/>redirect" type="hidden" value="<%= redirect %>
" />

</form>

<%@ include file="navigation.jspf" %>

</div>
```
References

[6] JISC. Shibboleth - connecting people to resources. 2.