Murad Benjadid Tanmoy

Single Sign-OnFeature For Customer Life-Cycle Management Application

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Techlology

Thesis

5 May 2017

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Murad BenjadidTanmoy Single Sign-On (SSO) Feature For Customer Life-Cycle Management Application 29 pages 5 May 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	TeroNurminen, Principal Lecturer

Signing into an application is the most critical part of any application, especially for an enterprise business application that needs to handle critical and highly sensitive user Information. An application like “SELFCARE”, which is thenewest and most recent product from Tecnotree Corporation must guarantee information security to its customer**s** before delivering the product.

However added security along with the immensecomplexity that comes with large-scale enterprise business application**s** can make signinginto an application very cumbersome especially in this case because the project application depends on a number of other application**s** to get its data to work .The main goal of the thesis **was** to find the best way to implement an architecture for singing into the application without sacrificing any security.

The SSO feature was successfullyimplementedas an architecture for signing in to the project application. After **i**mplementation of the feature it showed strong evidence that it highly improved the usability of the application. A number of penetration test**s**wereconducted by the security analyst to find any vulnerability of the implemented architecture. No security flaws were reported, whichproves the architecture has excellent security. The project application wasdelivered as a product to its customer in Iran in July 2016. Currently the application is used by millions of user**s**, with no complaints about the security and sign-on features. Aninitial report from the customer shows the product is a success.

Keywords	SSO,JSON,API,OAuth2, Puppet, SAML

Contents

1 Introduction

Nowadays competition to survive and generate revenue for software companies in the high stake digital market is immense. Software companies are not only required to create high quality software but also to integrate all the latest technologies in their products. Customers are becoming more and more demanding and always asking for the latest technologies for their products. One of the most recent trends for web based application is to feature single centralized login feature. And SSO based authentication provides most technologically advanced solution in this field.

The case company of this project is Tecnotree Corporation which is a telecommunication BSS (Business Support Solution) company founded in 1978 in Espoo Finland. The company was previously known as Tecnoman. In 2008 it acquired Indian Telecommunications Company Life Tree Coverage for USD46 million dollars and thus abbreviated to Tecnotree. Unfortunately, due to decline profit the company was subjected to bankruptcy in 2010. However, the company started to make slow comeback from 2012, and by 2015 its net sales reached to USD80 million. This success is partly due to providing the latest technology based software for its clients.

In 2015 the company successfully negotiated a contract with the MTN group, a multinational telecommunications company based in South-Africa to provide an application to manage mobile and other subscriptions provide by the MTN group. Surveys and research by MTN shows they spend a huge amount of money for setting up call and service centers for their customers, to troubleshoot customer complaints. If they can provide an application through which customers can manage their own subscriptionsthey can reduce huge load from their call centers and thus they can save money.

According to the contract the application should have all the latest technologies and the application should be highly user friendly and very easy to use. Since the SSO feature is the latest in the field of authentication technology, it was decided to implement this feature in the application. The application was named “SELF CARE” and I was assigned to implement the user authentication feature for the application.

The goal and objective of this thesis to introduce token based SSO authentication, steps, proceduresand to successfully implement a SSO architecture based application

. Any future company application can use this thesis asa guide, if they decided to use the same authentication architecture.

2 Introduction to SSO and JSON Web Token (JWT)

Authentication means verifying user credentials and establishing user identity whereas authorization deals with access control to certain allocated resources. SSO is strictly an authentication mechanism, it only deals with establishing an identity of the user and then sharing that information with the subsystem. Usually the Lightweight Directory Access Protocol (LDAP) and stored LDAP databases on stored served is used for the process. [6] SSO can be defied as property of access control and using this property user can be authenticated to multiple connected system or systems without the need to provide different user credentials. [7]

The SSO authentication is designed to address the issue of cross domain authentication. For example, if a web application is developed in domain A and developers decided to deploy the application to domain B, then how to authenticate the users for domain B who are already logged into to domain A. In this case,a session sharing solution cannot be implemented across cross domain due to browser's *same origin policy*.[6] The same origin policy restricts how resources originating form one origin interactwith resources that are originatefrom adifferent origin. That means cross domain information sharing is not possible. [8]

Let us assume that the URL for domain A is http://selfcare.company.com/dir/page.htmlIt will fail to share session information if the application is deployed to domain B if the URL for the domain B is such,https://selfcare.company.com/secure.html*here the protocol is different,*http://selfcare.company.com:81/dir/etc.html*here the port is different*,http://espoo.company.com/dir/other.html*here the host is different*.

There are several different SSO protocols namely Open ID Connect, SAML, Facebook Connect, Microsoft Account etc. I have used SAML SSO protocol for this application. The reason for using SAML protocol is discussed in sectionr4.2 under Supported protocols. The central concept for all this protocol is basically the same, that is a central domain or authentication server is used for authentication. The authentication server is

responsible for the generation of JSON (JavaScript Object Notation Syntax) web token which is unique and encrypted with JWE, JWE stands for JSON Web Encryption it encrypts resources with JSON-based cryptographic algorithm. This token cannot be altered and usually have a time limit for its validity after it expires, and no longer cannot be used as an authentication token. [9]

JSON Web Token (JWT) is a standard for passing JSON encoded information securely between two parties. The reason why JWT has gained so much popularity recently is that it can be transmitted by query, header and with the body of the post request due to its very compact size, enabling developer to develop an architecture totally based on APIs (Application Programming Interface). [10]Other benefits of using tokens over traditional methods are:

A. Scalability: Since the token contains all the necessary information's for authentication it frees server from storing any session state, thus it is stateless and can scaled very easily.

B. Ease of generation of tokens: Since token generation is decoupled from token verification it enables the user to manage the signing of tokens in different servers or to use different compiles like Auth0 for the purpose. [10]

C. Greater access control: A token can carry a payload which may contain all the user roles and permissions and as well as resources the user is allowed to access thus enabling greater access control

The token can be passed to the client and other associated domains for authentication. The token contains all the information to correctly identify and verify the user and his/her credentials.

3 Architecture

I have designed the architecture as such that the application communicates with the SSO serverusing standard **OAuth2/openid connect** or **SAML** protocol, WSO2 identity server was used here as SSO identity server. WSO2 is a third-party product, the detail description of its functionality is beyond the scope of this thesis however a product description can be found in reference [1]. The protocols define how SSO server can be utilized to obtain tokens representing user **authentication** and **authorization**. In addition, the clients use the SSO server to access **user profile information** and **token verification**.

By default, WSO2 is connected to TAP's (virtual *network kernel also known as network tap [11]*) internal LDAP. Any number of external LDAP (Light Weight Directory Access Protocol) or AD (Active Directory) servers can be configured as additional user stores. AD is a Microsoft specific directory service it supports a number of other protocols along with LDAP protocols to query data. Due to customer request, I made the decision to keep latest one AD server, so that the client could access data through Microsoft technologies

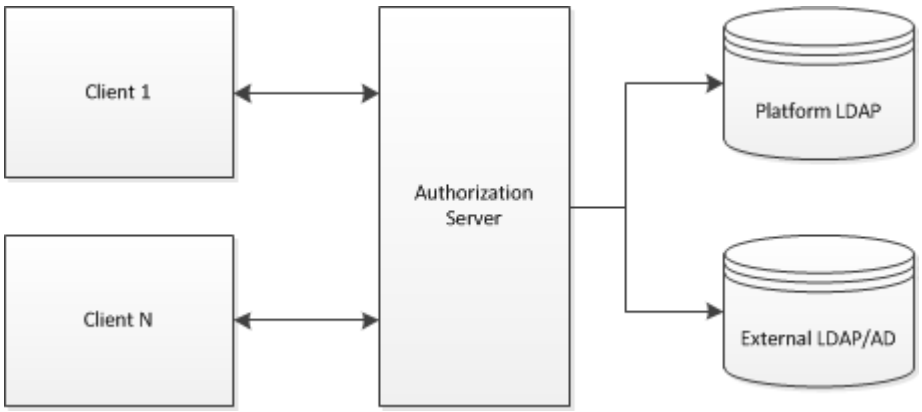


Figure 1, SSO architecture

Figure 1 shows us the basic functionality of SSO architecture, here the client first makes an authorization request to the authorization server, to get any LDAP service the client is required to get authenticated first. The authorization server authorizes the client after the authentication is done by WSO2 to access LDAP services. LDAP by default requires the information about who is accessing the data, so that it can decide whether the client is allowed to see the data. Only after the successful authentication and authorization of

the client , the LDAP server receives the actual client request and response with data. A simple request from the client involves sending username and password, since the password is received in clear text there is a security concern, because the password remains in the network and can be read from the network. I decided to use SSL (Secure Sockets Layer) encrypted channel for such operation. [12] The reason I decided to use SSL is because over the year SSL has been standardized as a standard security protocol and LDAP servers by default supports SSL protocol functionality.

Commonly the System Server will be used to host SSO functionality. A dedicated server can be used also if necessary. SSO can be installed on two System Servers and their databases are replicated. SSO server can be accessed from both System servers and through VIP or VIPA (Virtua IP address [13]). However, if System Server's VIP is changed the session will be lost, since no session data has been saved to the server, making the architecture totally stateless and highly scalable according to user needs.

SSO Single Sign On (SSO) is very practical solution to handle complex authentication feature for modern applications. Here the SSO architecture uses dynamic tokens, instead to session data.Tokens contains all the necessary information's to authenticate a user. However for this application I am also using a third party service from WSO2. The architecture also utilizes the *Puppet* framework. Puppet provides us the user Interface to to operate the software. And *Manaport* is actually our custom user interface that we build upon the puppet framework.

4 Supported Protocols

4.1 OAuth2

The legacy systems were already using OAuth version 1.0 for authentication so I decided to use OAuth2.0 to reduce development time and to remove any complexity for the implementation of authentication feature. OAuth2 is an open authorization protocol published in 2012. It is based on OAuth protocol created in late 2006. Currently OAuth2 is used by large internet companies such as Google, Facebook, Yahoo and Microsoft. For more information see reference. [2]

OAuth2 acts a resource owner that means it has the capability of granting access to protected resource. This is usually the end-user. The servers hosting protected resources can accept and responding to protected resources via request using access tokens , since the entire architecture is based on access tokens this is very important for the functionality of the entire application. Using the OAuth 2 protocol the server issues access tokens to the client after successfully authenticating the resource owner and obtaining the authorization. Here the client represents the entire application not just an individual user from server, desktop or other device's. The client (the application) makes the protected resource request on behalf of the resource owner and with its authorization. Also, I want to mention that OAuth2 now supports the *three-legged authentication,* this is where authorization and access token both can be done using OAuth2. This feature is not supported by OAuth version 1.0a which supports only so called *two-legged authentication where* the server is first assured to user identity before it can use access tokens. So by using SSL and OAuth 2.0 I was able to remove the need the for argument sorting and signing altogether ,now the client can simply pass its secret to the server which directly validates it. The OAuth version 2.0 also provide a new functionality called **refresh token.**

refresh tokens can be used as an equivalent of permanent password, so when the server needs access to protected resources it exchanged short lived access tokens over refresh tokens. Usually the expiration time for this short-lived access tokens is 5 minute.

In general, the protocol initiates when the resource owner authenticates itself to the authorization server. The resulting grant represents resource owner's authorization to access its protected resources. Clients who have access to a grant may exchange it for an access token. This token allows direct access to the protected resources.

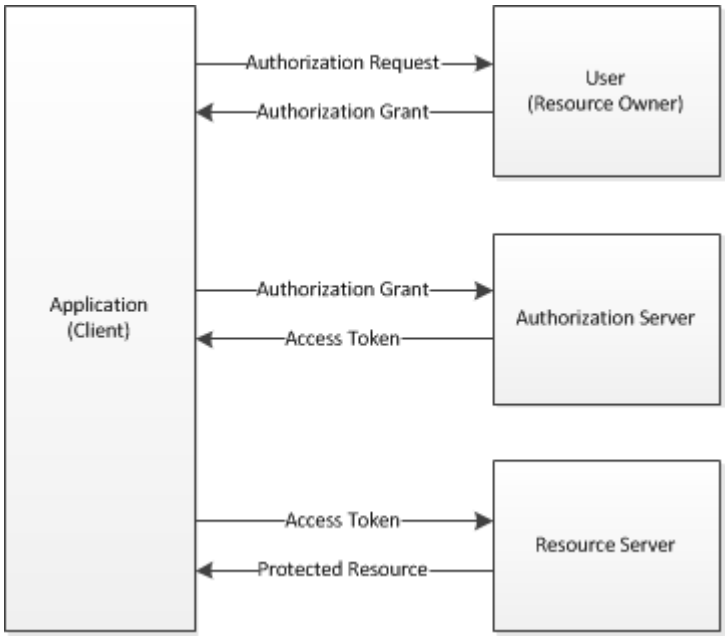


Figure 2, OAuth2 abstract work flow

Figure 2 illustrates the work flow of OAuth2. Here whenever Client makes any request the OAuth2 always perform three distinct role they are **Resource Owner, Authorization Server** and **Resource Server.**When client makes an authorization request, it grants authorization by acting as a resource owner, after the authorization is verified by the authorization server it issues access token to the client. Client uses this access token to get access to protected resources. OAuth 2 acts as a resource server after getting access token form the client and responds with protected resources. The main reason I decided to use OAuth over other protocol is because of its simplicity and reduced compellability. Most protocols usually require sighing procedure which is not necessarily very difficult not, but the step is quite complicated. A simpler and efficient protocols mean client can easily connect to the server service making the application to work faster, this is huge benefit if the application is deployed in a low bandwidth internet area. Since the product is intended for Iranian market, were internet bandwidth is very low and usually internet is also very expensive this protocol provides additional benefits.

The abstract flow is divided in three phases. The first one ensures that the client has permission from resource owner to act on its behalf. In second phase this permission is exchanged into a temporary token. In the third part the token is used to access protected resources.

The client implementations vary and have different requirements the OAuth2 defines four authorization grant types. Each grant is essentially the result of successfully completing the first phase of the protocol. The way a grant is obtained depends on its type.

Table 1 Additional roles of OAuth 2

Grant Type	**Description**
Authorization Code	Grant is obtained by directing the resource owner to the authorization server. After successful authentication, the resource owner is directed back to the client with the authorization code. Client can then exchange the grant to an access token with the authorization server.
Implicit Grant	Simplified version of authorization code designed for in-browser clients. In this flow the access token is issued directly after authentication instead of requiring the client to swap authorization code for it. This simplification reduces the number of round trips required to obtain an access token. However, is also means that the authorization server cannot authenticate the client which initiated the flow.
Password Grant	Resource owner's credentials can be used directly as an authorization grant to obtain an access token. This requires that the credentials are exposed to the client. This is acceptable only if there is a high degree of trust between the user and the client (e.g. a desktop program) or when other grant types are not available.
Client Credentials	Clients may use their own credentials as an authorization grant when the scope is limited to the protected resources under the control of the client.

Table 1 describes other additional roles and benefits of using OAuth 2 besides the main functionality which were mentioned earlier. OAuth2 protocol defines two endpoints that the authorization server must implement. Authorization endpoint is used to

initiate the protocol and to obtain a grant. Token endpoint is to exchange grants to access tokens.

.

The requests made to these endpoints differ based on which grant type is used. Grant types ***password*** and ***client credentials*** involve only the token endpoint. The third phase of the protocol is not defined fully in OAuth2 specification. Access token must be used to access the protected resource, but possible interaction between the authorization server and the resource server, the token validation is performed by OAuth2.

4.2 SAML

Security Assertion Markup Language or SAML is an authorization protocol commonly used in enterprise applications such as SAP and Salesforce. [14] The TAP provided SSO implementation supports SAML and applications may choose to use it. I have used SAML in this project for two reasons. These are:

A Standard:

The SAML standardized formatted is designed to be able to integrate with any system regardless of their implementation. Thus, developers can approach a more flexible and open architecture without thinking about vendor specific compatibility issues.

B Security:

The projection application deals with very sensitive user information details and information security is the most information factor for this project. SAML grantees a secure point single point of authentication. By providing a single point of authentication it ensures that user credentials never left the firewall boundary.

So, the application does not need to store any identities, SAML does this providing a strong security layer called the leverage public key (PKI) which also protects attack against assert identities Further description of SAML is beyond the scope of this document. [14]

5 Application Integration

5.1 Enabling SSO Functionality

This chapter describes the integration and deployment procedure of the system. After the initial deployment the access configuration is done via Manaport (*Tecnotree specific user interface for the identity server*) and add SSO template for the system server. The SSO template is found from **Additional templates** list on node metadata editor in the Manaport. After that I made all the necessary changes for the additional templates, then I deployed the system server to start SSO service. The SSO deployment configures the SSO to use the System Server's internal LDAP as authentication source. The usersrequired by the application must be inserted to the LDAP database on System server. This can be done via Manaport user manager or from the command line.

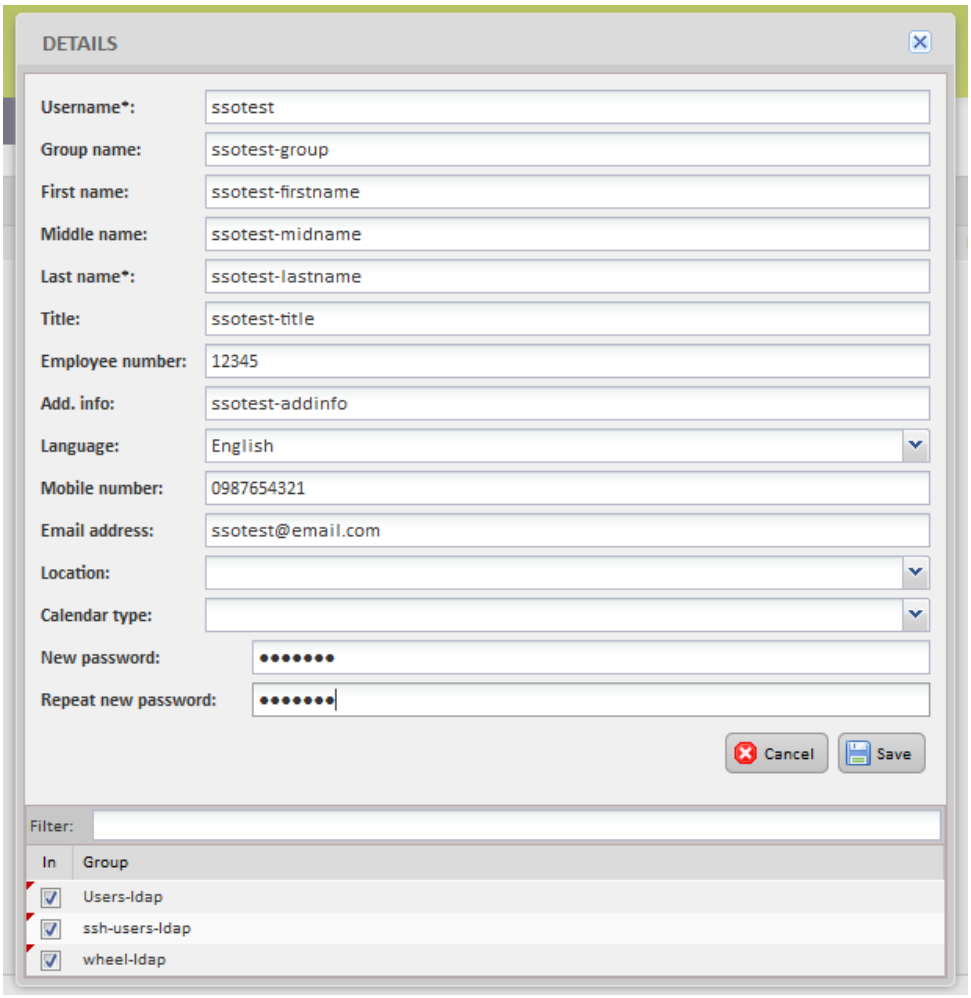


Figure 3, Manaport User Interface

Figure 3 shows how to create a test a user for the application using Manaport user interface. After the completion user receives a unique username and password. Manaport is only for developers, admins and application testers. The use of user interface greatly reduces time and fatigue of creating new user, deleting user and updating existing user. The same procedure can be done using console but that requires more time and prone to bugs.

To verify that the functionality is available, we need to login to the WSO2 admin console at: https://<SS IP>:9443/carbon. Here */carbon* is Manaport specific it opens up the admin user interface and SS IP depends on which server the application is running. The default credentials are **username:admin and password:admin**. After providing the credentials the Manaports redirects to user creation page, from here a new user can be created.

5.1 Service Registration

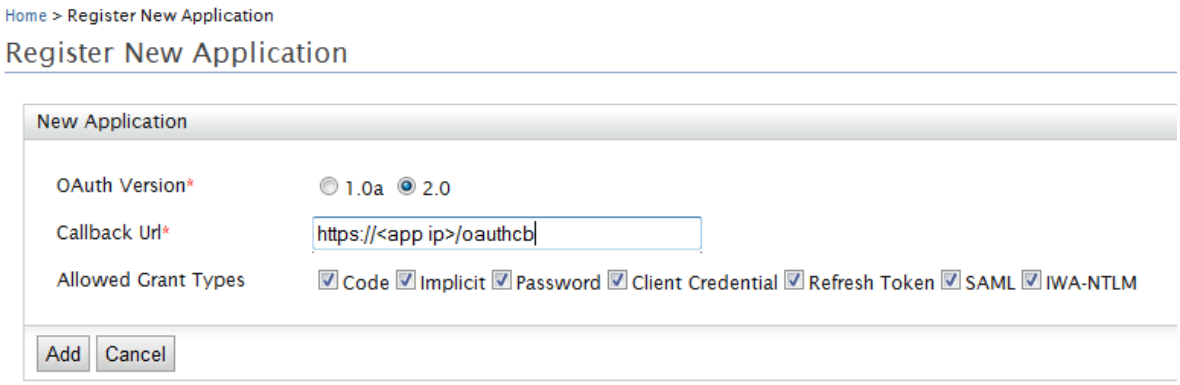
Each SSO service (or application) must register to the SSO server before anything can start. The server will assign credentials for the app. These are used to identify the source of incoming authentication requests. The registration process can be completed either manually via Web GUI or using a SOAP API. I find it easy to use Web GUI since it prompts messages if invalid information's are provided during the registration process on the other hand the SOAP(Simple Object Access Protocol) API crashes after if some information is in valid after the registration process, which is very annoying to my consideration .

5.1.1 Manual Registration

Manual registration is only for admins and developers testing the system, during the development and testing phase of the application, it was needed many times to try to register into and remove a user from the system and manual registration was used for the above situation. To perform manual registration admin into login to the WSO2 admin interface at https://<SS IP>:9443/carbon with credentials admin/admin.From the Main –tab admins add a new service provider and fill in the name of the application and a short description.

Next the admin needs to create application specific OAuth2 configuration under *inbound authentication configuration ->OAuth/OpenID Connect Configuration*. This part is required a callback address that SSO server uses to direct requests back to the application.

Since I was using JavaScript based clients(Application), the implicit grant type this is the same URL the application is served from. The callback address is expected to be https only. Https is not enforced but using plain http will leak tokens to anyone capable to monitor traffic between the authorization server and the client.



Registration is now complete. Client credentials are visible under OAuth2 configuration.

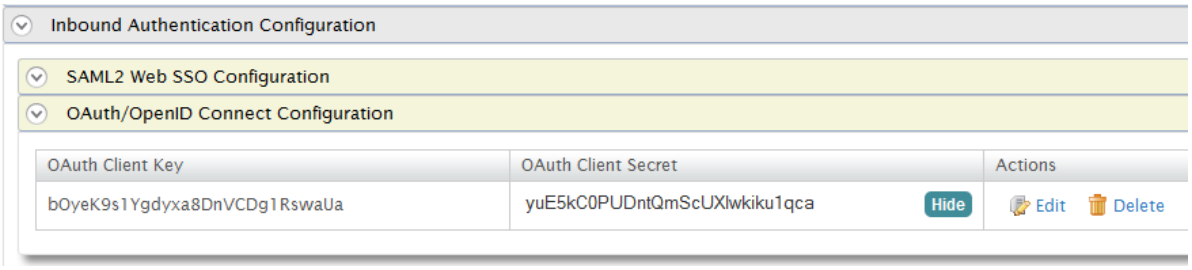


Figure 4, OAuth2 client registration user interface

Figure 4 shows the user interface provided by OAuth 2 for the client registration. Once the registration is completed, the client credentials are visible under OAuth2 configuration The ID and secret must then be made available for the client.

5.1.2 Registration via SOAP

The WSO2 Identity Server provides a SOAP API which I used for service/application registration, this is the ideal and recommended registration procedure. The SOAP API is called **OAuthAdminService.**

The SOAP calls can be done with HTTP Basic authentication with provided user-name/pw credentials. To register the following I made API calls, using the ***OAuthAdminService***.Then I register the application with ***registerOAuthApplicationData***. The registration data needs application name, callback URL for the application, allowed grant types *here I provided authorization code,* OAuth version which is 2.0. After that I generated **Application key** and **Secret key** using WSO2 identity server and I used them for the authentication process which done by using ***getOAuthApplicationDataName*** SOAP call. To make the API call I had to provide application name as parameter for the GET request. The API response carries the **OAuthConsumerKey and OAuthConsumerSecret**

Listing 1 Request and response format of SOAP API call

```
<soapenv:Envelopexmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://org.apache.axis2/xsd"
xmlns:xsd1="http://dto.oauth.identity.carbon.wso2.org/xsd">
<soapenv:Header/>
<soapenv:Body>
<xsd:registerOAuthApplicationData>
<xsd:application>
<xsd1:OAuthVersion>Oauth-2.0</xsd1:OAuthVersion>
<xsd1:applicationName>TestApp</xsd1:applicationName>
<xsd1:callbackUrl>https://<host>/<callback></xsd1:callbackUrl>
<xsd1:grantTypes>authorization_code implicit</xsd1:grantTypes>
</xsd:application>
</xsd:registerOAuthApplicationData>
</soapenv:Body>
</soapenv:Envelope>
```

The **getOAuthApplicationDataByName** response:

```
<soapenv:Envelopexmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<ns:getOAuthApplicationDataByAppNameResponsexmlns:ns="http://org.apache.axis2/
xsd">
<ns:returnxsi:type="ax2324:OAuthConsumerAppDTO"
xmlns:ax2324="http://dto.oauth.identity.carbon.wso2.org/xsd"
xmlns:ax2326="http://oauth.identity.carbon.wso2.org/xsd" xmlns:xsi="
```

Listing 1 shows how the standard SOAP calls are made, here I used Java based API calls. This section describes the authentication procedure clients need to implement. The implementation depends on which grant types the clients choose to support. The recommended grant type is ***authorization code***. Other grant types may be used only if authorization code is not suitable to the use case.

All HTTP requests in this section are made using ***application/x-www-form-urlencoded*** format unless stated otherwise. The token response is similar for each grant type. It is described in more detail in section5.4

5.1.3 Authorization Code Grant

This section describes in detail the functionality and format of the authorization code, authorization code is the recommended grant type for server side applications. Client initiates the login flow by sending a request to authorization endpoint.

https://<SSO IP>:9443/oauth2/authorize

Parameter	**Value**
client_id	Client specific OAuth2 ID obtained when the application was registered to the SSO server
client_secret	Client specific OAuth2 key obtained when the application was registered to the SSO server
response_type	Fixed to code. This initiates authorization code flow.
redirect_uri	The callback URL defined during the application registration
scope	Fixed to openid. SSO server will not reply to user profile requests if this is not set.
state	Optional. A client-determined value used to maintain state between the request and the callback. This value is included in the server response when redirecting the user-agent back to the client.

. Table 2 Request parameters for authorization endpoint

Table 2 describes the necessary parameters required for a successful request to authorization endpoint, if it fails the server responds with 404 Not Found status code. Next steps of authorization are performed by the SSO server. First the user-agent is

directed to the generic login page. After successful login SSO server generates the grant and redirects the user-agent to the application callback.

The callback must extract the authorization code from the server response. The code can then be exchanged for an access token utilizing the token endpoint:

https://<SSO IP>:9443/oauth2/token

Parameter	**Value**
client_id	Client specific OAuth2 ID obtained when the application was registered to the SSO server
client_secret	Client specific OAuth2 key obtained when the application was registered to the SSO server
redirect_uri	The callback URL defined during the application registration
grant_type	Set to authorization code.
code	Received authorization code

Table 3 Request parameters for token endpoint.

Table 3 describes the parameters needed to make any request to token endpoint. The token endpoint validates the client credentials and the authorization code. If valid, the authorization server responds with access tokens and with HTTP status code 200. However, if the operation fails it sends 405 **Method Not Found** status code.

5.1.4 Implicit Grant

The implicit Grant is the most important authentication flow for this application, since most of the user for this application is expected to be browser based. Implicit Grant flow is suitable for in-browser clients. Login flow with implicit grant does not involve temporary authorization code. Instead the access token is given directly to the client after successful authentication. This grant type does not include client authentication because client credentials can't be hidden from the user. Furthermore, the access token is encoded into the redirection URI which may expose it to the user and other applications running on the same device. Application initiates the login flow with a request to authorization endpoint.

https://<SSO IP>:9443/oauth2/authorize

The request must have following parameters, must have *client_id, response_type, redirect_uri,scope and state*. Here state is optional, A client-determined value used to maintain state between the request and the callback. This value is included in the server response when redirecting the *user-agent* back to the client

Next the user-agent is directed to the generic login page. After successful login SSO server generates the token and directs it to the application callback using URL fragments. The client is expected to parse the token from the response.

5.1.5 Password Grant

Password grant is the most easy to use authentication flow, the reason why I used this authentication flow is to help the application testing team. This authentication flow is suitable for test automation and unit testing of registration of a user. However it is not recommended for applications at all because the user credentials are exposed to the application. Any data that is present in the client side of the application is prone to security risk. The flow consists of a single request to the token endpoint:

https://<SSO IP>:9443/oauth2/token

The request parameters are same as the other authentication request, except for two additional parameters. These are username and password which are owner's name and owner's password respectively. SSO server responds directly with the access token. This flow does not involve user-agent redirections. The request is a POST request and if successful the server responds with HTTP 200 OK response.

5.1.6 Client Credentials

Client credentials can be used in situations where the client needs to access resources which are not tied to a specific resource owner. The flow consists of a single request to the token endpoint:

https://<SSO IP>:9443/oauth2/token

The request is made with two additional parameters, client secret (client specific OAuth2 Id is obtained when the application is registered to the SSO server) and grant

type which is fixed to client credentials. The request is a post request and if successful the status code is 200 OK.

5.2 OAuth2 Token Response Content

This section describes the content of the response after successful request had been made. The response from token endpoint contains the access token issued by the server the token type defines what kind of token was returned. Currently only bearer tokens are supported. The response token has also a time limit which can be found under parameter ***expires_in*** and it is usually in millisecond format. I had to convert it every time during the development of the application. The refresh token is optional it is additional token that can be used to renew the access token. Refresh token is not included in implicit grant flow. The scope parameters contain the scope for which the token was issued. In practice this is always fixed to *openid*.Along with refresh token.***id_token*** is also an optional parameter.

5.2.1 OpenID Connect Token

Access tokens using **openid** scope are accompanied with JSON Web Token.The token is base64 encoded. The data is in three parts separated with dots. The first part defines the signature algorithm. The second part contains the claims and the third part contains the signature

Claim	**Name**	**Explanation**
iss	Issuer claim	The iss (issuer) claim identifies the principal that issued the JWT. The processing of this claim is generally application specific. OPTIONAL
sub	Subject claim	The sub (subject) claim identifies the principal that is the subject of the JWT. The Claims in a JWT are normally statements about the subject. OPTIONAL
aud	Audience claim	The aud (audience) claim identifies the recipients that the JWT is intended for. OPTIONAL
exp	Expiration time claim	The exp (expiration time) claim identifies the expiration time on or after which the JWT MUST NOT be accepted for processing. OPTIONAL
nbf	Not before claim	The nbf (not before) claim identifies the time before which the JWT MUST NOT be accepted for

Claim	Name	Explanation
		processing. OPTIONAL
iat	Issued at claim	The iat (issued at) claim identifies the time at which the JWT was issued. This claim can be used to determine the age of the JWT. OPTIONAL
jti	JWT ID claim	The jti (JWT ID) claim provides a unique identifier for the JWT. OPTIONAL

Table 4 Token content [4].

Table 4 describes different fields that are present in a decoded token. All this information is necessary for successful authentication, and token's ability to carry all this information in encoded format insures not only security but also efficiency.

5.3 Operations with Valid Tokens

The architecture uses token based authentication instead of using session base user authentication to improve security. Unlike session based authentication the authentication tokens are not present on the on the browser making the application more secure. The SSO server generates new access token with every new request with predated time of its validity. The validation time can be set to any but its recommend and used in this project as 30 minutes after that the token expires and send failure JSON response with 401 status code.

Resource servers are expected to validate the token before serving the resources.Clients must include the access token in every request to protected resources. The token is set in the authorization header, the token type must be capitalized when making requests to the authorization server. For example:

Authorization: Bearer d11826442326856d5e9747ed777c8f11

5.3.1 Obtaining User Profile

User profile information is available in**userinfo endpoint**.*https://<sso ip>:9443/oauth2/userinfo?schema=openid.*The endpoint supports only **openid** schema. Requests must include an authorization header with a valid access token. This access token must have openid scope.

5.3.2 Permission Handling

Manaport is an unique product of Tecnotree , it is basically only an user interface the user info must come below format to access the functionality Manaport. The wildcard (*) at the end of permission string denotes all possible permissions after it. It is the applications responsibility to expand the wildcard operation and act accordingly. If there are no permissions, the full permission entry may be missing. The permissions may also appear in the string in any order.The wildcarded permissions may overlap with each other. For example, it is possible the permission set has *“platform.*,platform.alarms,*”* The user info as requested from the **userinfo** endpoint contains the user’s authorisation information in form of permissions, as follows:

”permissions”: ”<comma separated permission list>”,

5.3.3 Token Validation

The authorization server does not provide an endpoint for simply determining the status of given token. For now, I have decided that the user info endpoint (described in the previous section) is used for this purpose, if needed the endpoint can easily be configured. The user profile is available only when presented with a valid access token. An invalid token will return with HTTP 400 error code. Token format is depended on the protocols. SAML protocols only supports SAML tokens and OAuth2 supports JWT based tokens. Here the authorization protocols are only concerned about the token format the application doest not need to care about the formats of the tokens and how they are handled. The mechanics of token validation involves well-formed tokens, tokens are coming from correct and indented authority and after the token are verified they are meant for the correct application. Well-formed tokens mean nobody has tam-

pered with the tokens, the tokens are received before they have been expired and it is correctly formatted containing all the necessary fields. Tokens are designed in such a way that they clearly display information about their origin and they are unambiguous as much possible. This is to Identify the Intended authority easily.

5.3.4 Token Refresh

Token Refresh is a very special type of token supported by OAuth to authenticate client. Usually the user needs to get a new token very time the tokens expires and then re-authenticate to the system but with refresh token user does not need to re-authenticate. It acts a permanent token similar to passwords provided by user. They are received via OAuth API. In the OAuth dashboard, a refresh token is clearly visible and from here the admin can revoke re re-issue a refresh token. To get a refresh token authentication request must go through the authorize end and it is required to send device name as parameter within the request. The device name can have any value for example a valid mobile number can be used as a device name. Since there is no time limit for the expiration of a refresh token, it is very important to revoke them time to time. OAuth has a management API to revoke a refresh token.

The revoke request needs to send id of the refresh token to be revoked. The request is DELETE CRUD operation and if successful the server responded with 204 status code. And the credentials are no longer exists in the server,Authorization server issues new refresh tokens along with access tokens. The refresh tokens can be used to obtain a new access token when the current one has expired. OAuth also have management API dashboard user interface, which provides the easiest option to revoke refresh tokens. But the API is much more programmable. I have programmed the authentication as such that it calls the management API every 24 hour to revoke the refresh token. This ensures enhanced security.

5.3.5 Token Revocation

Access tokens and refresh tokens can be revoked. OAuth 2 guidelines suggests that every implementation of OAuth must have an access token revocation functionality. [15] A token revocation request is made at the revocation end point URL, it is a POST request and contains the id of the access token to be revoked. I have programmed the application as such that submitting any revocation request the server first need to validate the client information to authenticate the client and to check if he have rights to perform such operation.

Revocation request must contain the token and its type. Requests must be sent with an authorization header containing a valid access token or with basic OAuth using client credentials.After revocation attempts to access protected resources must fail due to invalid access token

5.4 HTTPS Certificates

All operations with authorization server must be secure. The server certificates in TAP 2.3 / TAP 2.4 releases are pre-generated with a fixed common name. This will cause hostname validation errors when forming https connections.Server certificates are stored in a JKS file at path:

/opt/wso2/repository/resources/security/wso2carbon.jks

Keystore password is **wso2carbon**. Command ***keytool*** can be used to interact with the file. The key entry for the server is ***wso2carbon*** and the certificate is ***wso2carbon.cert***.

5.5 SCIM Provisioning Interface

SCIM stands for System for Cross-domain Identity Management, it was designed to manage identities of cloud based applications, SCIM provides the vital cross domain functionality for the SSO architecture that this thesis paper thrives to achieve. SCIM interface can be used to create, modify and delete user and group information. The

interface is enabled by default. All requests to the interface must be done with credentials of an admin user or with credentials of a service provider (OAuth client key/secret work).

SCIM interface can operate with any user store where the SSO server has write access. The target user store must be specified in all operations by prefixing the user/group name with the user store ID. The default user store ID for LDAP located on System Server is ROOT. Any operation targeting the LDAP must then specify the resource names endpoints. SCIM is based on an object model where resource is the main denominator and all the SCIM objects are based on it. [16] In SCIM is encoded and represented as an JSON object that has id, username etc. All the attributes have string value. SCIM also uses group to give an organizational structure to the provisioned resources, to manipulate this resources SCIM has a REST API that supports all the CRUD operations. SCIM is used to simplify user management in the cloud by defining a schema for representing user and groups [5] SCIM provides following endpoints:

- https://<SSO SERVER>:9443/wso2/scim/Users
- https://<SSO SERVER>:9443/wso2/scim/Groups
- https://<SSO SERVER>:9443/wso2/scim/Bulk

Operations targeting single users or groups are done via the resource type specific endpoint. Alternatively, bulk endpoint can be used for provisioning large sets of users or groups.

SCIM API is used by making HTTP requests to the above endpoints. The type of the request defines the API function (GET, POST, PUT, DELETE). The request must be authorized with a Bearer token or HTTP basic auth. The content type is application/json with function specific payload.In general, the API will return 200 OK and application/json data for a successful request. Modifications to existing resources are done with **replace** operation. SCIM specification defines an **update** operation, but WSO2 does not support it.

5.5.1 SCIM Schema Claims

SCIM supports a set of user/group attributes defines by the SCIM schema (urn:scim:schemas:core:1.0). The schema and its attributes are shown in the WSO2 GUI (Configure -> Claim Management).This mapping is used to configure which data

store attributes map to which API attributes. For example, when a new user is added her username is passed through the API as a value for urn:scim:schemas:core:1.0:userName. In LDAP the username value would be stored to uid attribute.By default the SCIM mapping is incomplete regarding the Platform LDAP. The following claims must be mapped manually.

Claim	MappedAttributes	Description
Id	scimid;ROOT/labeledUri	Unique ID of the SCIM resource
Meta - Created	createdDate;ROOT/carLicense	Creationtimestamp
Meta - LastModified	lastModifiedDate;ROOT/gecos	Modificationtimestamp
Meta - Location	l;ROOT/homePostalAddress	Unique URI for this SCIM resource. Accessing the URI will display the resources data

Table 5 SCIM schema.

Table 5 illustrates the claims that can be made using SCIM schema. WSO2 supports mapping attributes to multiple user stores. The user stores are separated with semicolons. The mapping is given as <User store ID>/<attribute>. In the above table the mappings refer to the internal WSO2 user store (no ID) and the Platform LDAP (ROOT/*).
The target LDAP attributes defined in the table work, but they are subject to change.
If the claims are not mapped properly SCIM API requests will likely dump Java exceptions.

5.5.2 Read and Search Operations

SCIM can be used to dump the data of any user or group. The API provides ways to access a single resource, all resources of given type or all resources that match a search query.Read and search requests must include an authorization header with valid bearer token or use basic auth.

5.5.3 Create Operation

To create a resource (user or group), send an HTTP POST request to the resource specific endpoint. The request must be authorized with a Bearer token or HTTP basic auth. The content type is application/jsonand it is a POST request. The URL of the request have a version number so that different versions of the SCIM API can be used at the same time. To check the available version SCIM provides a **ServicProviderConfig** end-point. The minimal payload can be extended by defining additional user/group attributes. The available attributes depend on the SCIM claim configuration. On success request the server responds with 201 status code.

The response contains the created the resources with unique id and meta data. The response also contains a location header to point where the resources can be fetched next. The service providers added meta data to make the resources complete.

5.5.4 Replace Operation

Replace operation is used to modify the attributes of existing user or group. The target data will be completely replaced with the data given in the request. It is essential that the request defines values for all attributes even if they do not change.Replace operation works for groups in the same way as for users.

5.5.5 Delete Operation

Delete operation removes the target user or group from the user store. This is done with http DELETE request to the unique URL of the target resource. I have decided not to permanently delete the resources due regulation concerns, but if DELETE request has been made on any user or group the server returns 404 error code without permanently deleting the user.

5.5.6 Bulk Requests

Bulk requests can be used to run a large set of SCIM operations using a single request. All bulk requests use the same endpoint regardless of whether they target users or groups. The bulk requests basically consist of a set of individual SCIM operations. BulkId for each operation must be unique. It is used to identify the operations and their response. BULK requests are optional operation provided by SCIM. Bulk request is only used if user decided to send a very large collection of resources using HTTP API methods.Other operations (read, search, replace) can be used in the bulk request in the same way. Attribute ***method*** controls which operation is called. If a bulk request is successful the server responds with code 200 OK.

5.5.7 Restrictions with SCIM

SCIM interface in the current builds suffers from several problems. SCIM can manage only those users/groups that have been created by SCIM, claim mapping configuration must be performed manually. Some attributes (permissions) are not available via the mapping.User passwords must be plaintext for SCIM API which is not ideal. Have raises serious security issue. The SCIM protocol is totally based on REST API and it is assumed that any application using SCIM uses restful APIs. But in practical situation this is not always true. It is very hard to create an application entirely using custom APsI. So, there will be always some third part API which are not Restful. These might cause the SCIM protocol to crash. During the development and testing phase I have not suffered such problems but it might happen and the application might needed to be upgraded with a different protocol to support cross-domain functionality.

The multi-valued attributes make resource manipulation very clumsy. SCIM unfortunately does not pay any attention to this issue which is very disappointing. Problem is mostly because every value need a unique key but multi-valued attributes seems to lack this key. The way I tried to solve this issue is by turning very list of arrays into a dictionary and providing each of them with a unique meaningful free identifier. SCIM is one of the first protocols to use PATCH HTTP verb. PATCH is used when it is needed to update a part of resource not the entire resource. PUT is used if we want to update the entire resource. PACTH allows part of a resource to be modified, added or deleted. When I used PATCH with SCIM protocol it looks very troublesome to me. One major issue that I had is that it always gives me 200 OK code for all PATCH requests. So I

have no way to know whether the right changes has been made. This is also very disappointing considering that this is very easy problem to fix but obviously lacks attention from SCIM developers to fix the Issue. I was forced to use PUT methods instead to PATCH to make minor changes to allocated resources. I hope they will soon address this issue and will improve their product.

6 Conclusion

In conclusion, the goal of the thesis was to build a working architecture for an application with SSO functionality with cross-domain compatibility. As a result of the project the application “SELF CARE” was produced that has the architecture to support all SSO functionality. Due to its cross-domain functionality the application can be deployed

at any domain at any time, without making any major changes to its backend logic. During development of this software we have tested all the SSO functionality by deploying the application in several virtual machines running in different ports. During the testing phase I have not noticed any major Issues, except some minor issues regarding SAML protocol.
I believe a better a protocol can be used in future instead to SAML to improve the performance of the overall system.

The application has already been delivered to our client in Iran, “SELF CARE” went live in August 2016. Currently the application is used by millions of MTN customers, it was a successful deployment to MTN’s own server. The initial user view suggest a successful application launch and introduction on SSO feature into the “SELF CARE” application enables a smooth transition of an old CLM application that MTN was using which was decommissioned after the new launch. The new selfcare application uses all the existing API from the decommissioned application thank to the SSO feature reducing huge amount of development and deployment costs.

MTN wanted this application to reduce the cost of operating their call and service centers, no data is yet available to suggest that the application has managed to achieve it’s goal of reducing costs. It will take more than two to three years for the application be stable, during this time we plan to improve the application by carefully going thorough test and customer user experience data. But I am quite hopeful, as more and more customers starts using our application and understands it’s benefits they will eventually become less depended to the call centers.

To summarize my work for this thesis project, I have successfully designed and implemented a Single Sing On (SSO) architecture for a very complex Application. I have tested the application during production and after deployment to a number of customer domains, the application and the SSO architecture performed perfectly, which is very satisfactory and encouraging for my future projects.

References

1 WSO2 Identity Server[online] URL:

http://www.peterjthomson.com/2009/08/difference-between-marketing-andde

sign/. Accessed January 1, 2016.

2. The OAuth 2.0 Authorization Framework [online] URL:

http://tools.ietf.org/html/rfc6749.Accessed January 1, 2016

3. WSO2 Identity Server[online] URL:

http://www.peterjthomson.com/2009/08/difference-between-marketing-andde

sign/.

Accessed January 3, 2016.

4 JSON Web Token [online] URL:

https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32. Accessed March 20, 2016.

5 Cross-domain Identity Management [online] URL:

https://www.simplecloud.info/.
Accessed April 3 2017

6 What is how does Single Sign On Authentication works [online] URL:

https://auth0.com/blog/what-is-and-how-does-single-sign-on-work.
Accessed April 25 2017

7 Single sign-on [Online] URL:
https://en.wikipedia.org/wiki/Single_sign-on .
Accessed April 25 2017

8 Same-origin policy [Online] URL:
https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy.
Accessed April 25 2017

9 JSON Web Encryption (JWE) [Online] URL:
https://tools.ietf.org/html/rfc7516.
Accessed April 25 2017

10 Token Based Authentication Made Easy [Online] URL:
https://auth0.com/learn/token-based-authentication-made-easy.
Accessed April 25 2017

11 TUN/TAP [Online] URL:
https://en.wikipedia.org/wiki/TUN/TAP.
Accessed May 1 2017

12 LDAP unix [Online] URL:
http://www.tldp.org/HOWTO/LDAP-HOWTO/authentication.html
Accessed April May1 2017

13 Virtual IP address [Online] URL:
https://en.wikipedia.org/wiki/Virtual_IP_address.
Accessed May1 25 2017

14 Dev Overview of SAML[Online] URL:
https://developers.onelogin.com/saml.
Accessed May 2 2017

15 Token Revocation [Online] URL:
https://tools.ietf.org/html/rfc7009.
Accessed May 2 2017

16 SCIM [Online] URL:
http://www.simplecloud.info/.
Accessed May 2 2017