Seam Social Module

Reference Guide

3.1.0-SNAPSHOT

by Antoine Sabot-Durand

Introduction v
1. Getting Started 1
1.1. Building 1
1.2. Usage big picture 1
1.3. Starting with OAuth configuration 2
1.3.1. Create an OAutConfigSettings bean thru Seam configuration (in bean.xml) 2
1.3.2. Adding the @ConfigureOAuth annotation when injecting the OAuth service bean 3
1.4. Inject an OAuthService bean with one of the following ways : 3
1.5. Request the Authorization URL for the service and redirect the app to this url 4
1.6. Store the verifier in OAuthService bean and init access token 4
1.7. After what we can send calls to the service 5
1.8. Testing 5

Introduction

The goal of Seam Social is to provide CDI components to access some of the social network services. It is focused on OAuth 1.0 and 2.0 services. At least Seam Social provides a way to get one's profile for a given service. There are services like Twitter for which Seam Social provides a lot of model object to manage timeline and friends in the service.

Seam social was created with genericity and polymorphism concepts in minds allowing users to deal with a collection of OAuth sessions, it also allows you to create your own extension to connect to an OAuth server.

This version of Seam Social provides support for the following OAuth Services with the following level

Table 1.

Service	Profile	Update	Timeline	Friends
Twitter	X	X	O	O
LinkedIn	X	O	O	O
Facebook	X	X	O	O

More services and level of support to come.

Getting Started

Provides CDI Beans and extensions to interact with major social network.

Provides:

- OAuth connectors to authentify with an OAuth providers
- Support for Authentication for Twitter, LinkedIn and Facebook only right now
- Status update for Facebook Twitter and LinkedIn
- Support for multi-account (multi-service and multi session for the same service)

Seam Social is independent of CDI implementation and fully portable between Java EE 6 and Servlet environments enhanced with CDI. It can be also used with CDI in JSE (desktop application). It is build on top of *scribe-java from fernandezpablo85* [https://github.com/fernandezpablo85/scribe-java]

For more information, see the *Seam Social project page* [http://seamframework.org/Seam3/Social].

1.1. Building

```
mvn -Pweld-ee-embedded-1.1 clean install
```

you need to be connected to internet to launch the tests. You can build without the tests like that :

```
mvn clean install -DskipTests=true
```

1.2. Usage big picture

The Web example app is quite simple and give a good idea of possibilities of Seam Social Framework.

Main steps to use Seam Social are :

- Declare an OAuth configuration
- Inject an OAuthService bean

- Request the Authorization URL for the service and get a request token
- Store the verifier in OAuthService bean and init access token
- Use the service

Should you need to fully understand each step, the complete OAuth lifecycle can be found *here* [https://dev.twitter.com/docs/auth/oauth] or *here* [https://developer.linkedin.com/documents/authentication]

1.3. Starting with OAuth configuration

To consume an OAuth service you need to declare an application on the service platform (i.e. for Twitter you can do, this on *https://dev.twitter.com/apps/new*). The declaration of an application contains at least :

- an API public key
- an API private/secret key

To use an OAuth service bean in Seam social you need to provide these configuration information in two ways :

- thru an OAuthConfigSettings bean
- by adding the @ConfigureOAuth annotation when injecting the OAuth service bean

1.3.1. Create an OAutConfigSettings bean thru Seam configuration (in bean.xml)

Right now, Seam Social provides only one convenient way to declare an OAuthConfigSettings bean. It can be done thru Seam configuration file (beans.xml). Here is an example of such a configuration :

```
<beans xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:s="urn:java:ee"
   xmlns:o="urn:java:org.jboss.seam.social.core">
   <o:OAuthServiceSettingsImpl>
      <s:modifies />
      <o:RelatedTo>Twitter</o:RelatedTo>
      <o:apiKey>FQzlQC49UhvbMZoxUIvHTQ</o:apiKey>
      <o:apiSecret>VQ5CZHG4qUoAkUUmckPn4iN4yyjBKcORTW0wnok4r1k
```

```
        </o:apiSecret>
        <o:callback>http://localhost:8080/social-web-client/callback.jsf
        </o:callback>
    </o:OAuthServiceSettingsImpl>
</beans>
```

Api Key and Api secret is provided by the service you want to consume (here Twitter). You can use the values above since they're coming from "Seam Social" Twitter application. Callback depends on your application : it's the URL that will collect OAuth verifier

1.3.2. Adding the @ConfigureOAuth annotation when injecting the OAuth service bean

You can simply add the @ConfigureOAuth annotation to the injection point. It can be done like that :

```
@Inject
@ConfigureOAuth(apiKey = "FQzlQC49UhvbMZoxUIvHTQ", apiSecret =
    "VQ5CZHG4qUoAkUUmckPn4iN4yyjBKcORTW0wnok4r1k", callback="http://localhost:8080/
social-web-client/callback.jsf")
Twitter twitter;
```

With this notation the injected bean is configured with the given OAuth values.

1.4. Inject an OAuthService bean with one of the following ways :

Using the Interface of the service

```
@Named
@SessionScoped
public class mySessionBean implements Serializable {
    ...
    @Inject
    public Twitter twitter;
    ...
}
```

or using the generic OAuthService with a Qualifier

```
@Named
@SessionScoped
public class mySessionBean implements Serializable {
    ...
    @Inject
    @RelatedTo("Twitter")
    OAuthService service;
    ...
}
```

The two are equivalent but the second one give you a way to do polymorphic calls to the service. The OAuthService provides methods in relation to authentication.

1.5. Request the Authorization URL for the service and redirect the app to this url

If we go on with the same example, we can get this authorization URL with this call :

```
twitter.getAuthorizationUrl();
```

It will return the URL needed to initiate connection to the service.

1.6. Store the verifier in OAuthService bean and init access token

When we return from the service connection to the callback URL, we get a verifier that we need to store in the OAuthService and init the access token In JSF we do this like that

```
<f:metadata>
    <f:viewParam name="#{mySessionBean.twitter.verifierParamName}"
        value="#{mySessionBean.twitter.verifier}"
                required="true"
                requiredMessage="Error with Twitter. Retry later"/>
    <f:event type="preRenderView"
        listener="#{mySessionBean.twitter.initAccessToken()}"/>
</f:metadata>
```

1.7. After what we can send calls to the service

Getting the Twitter user profile

```
TwitterProfile user = twitter.getMyProfile();
String fullName = user.getFullName();
```

1.8. Testing

After building you can deploy the war generated in example/web-client/target in a Java EE 6 container implementing web profile (tested with JBoss 6 but should work in glassfish too)