

HOSPITAL TRUETA

MANUAL ALFRESCO

Autor: IN2 – Ingeniería de la Información
Fecha: 01 de Agosto de 2017
Versión: 1.0

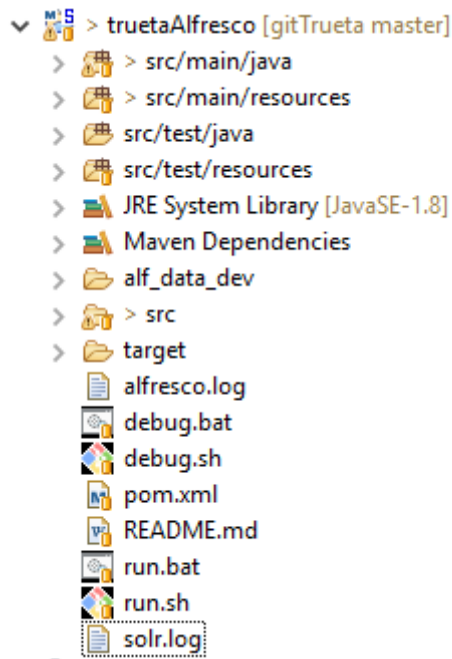
Tabla de contenidos

1	Estructura proyecto.....	3
1.1	Proyecto Alfresco	3
1.1.1	src/main/java	3
1.1.2	src/main/resources	3
1.1.3	src/test/java	5
1.1.4	src/test/resources	5
1.1.5	JRE System Library.....	6
1.1.6	Maven Dependencies	6
1.1.7	alf_data_dev.....	6
1.1.8	src	7
1.1.9	target.....	7
1.1.10	Resto de ficheros.....	8
1.2	Proyecto Share	8
1.2.1	src/main/java	9
1.2.2	src/main/resources	9
1.2.3	src/test/java	10
1.2.4	src/test/resources	10
1.2.5	JRE System Library.....	10
1.2.6	Maven dependencies	11
1.2.7	src	11
1.2.8	target.....	12
1.2.9	Resto de ficheros.....	12
2	Extensiones	13
2.1	Alfresco.....	13
2.1.1	Modelo	13
2.1.2	Action	14
2.1.3	Behaviour	17
2.1.4	WebScript.....	19
2.2	Share.....	21
2.2.1	Formularios	21

1 Estructura proyecto

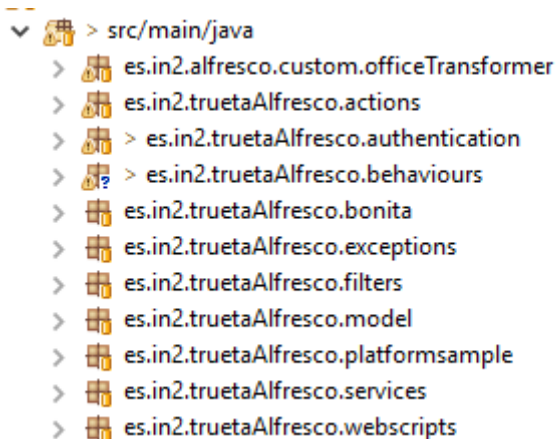
1.1 Proyecto Alfresco

La estructura del proyecto es la siguiente:



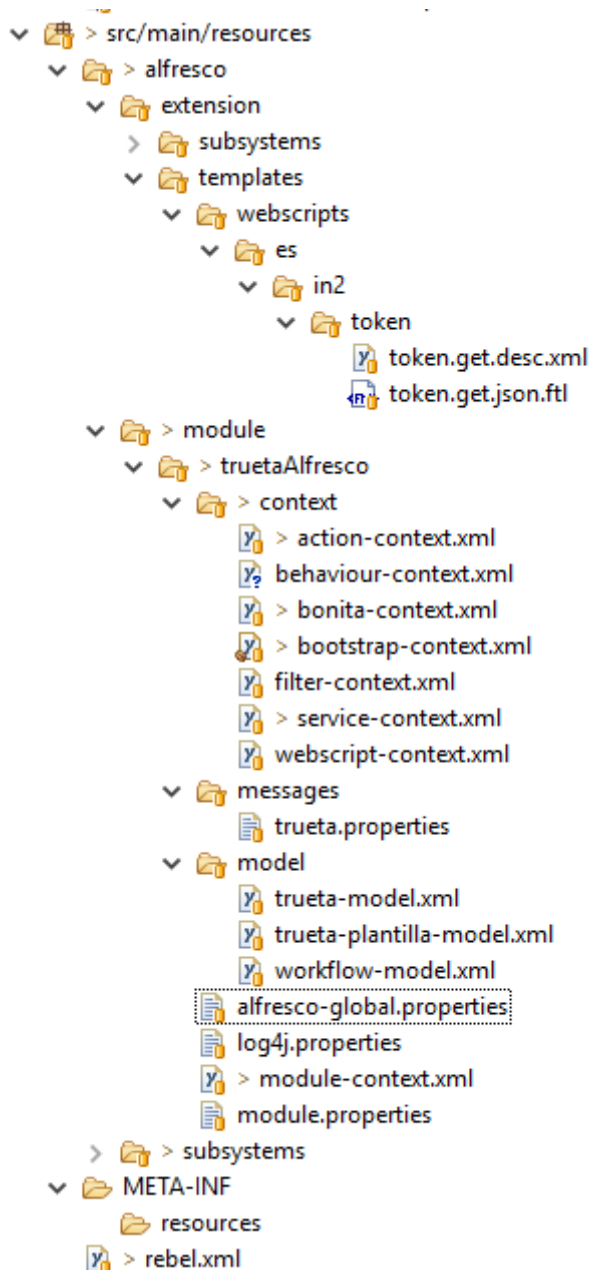
1.1.1 src/main/java

En la carpeta src/main/java tendremos los fuentes. La estructura de las carpetas es opcional pero es recomendable seguir un patrón que nos ayude a encontrar las clases.



1.1.2 src/main/resources

En la carpeta src/main/resources tendremos los ficheros de recursos, es decir, los ficheros que los fuentes necesitan.



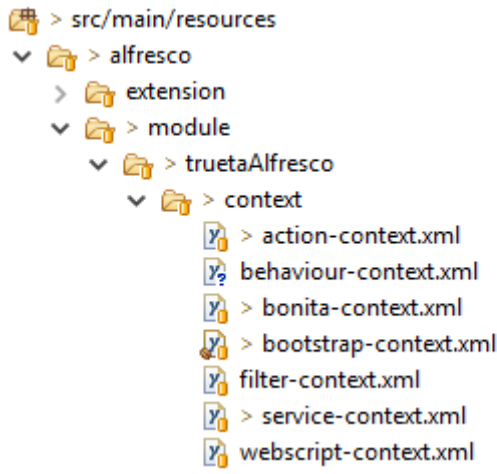
En el primer nivel encontramos:

1. alfresco
 - a. extensión
 - i. subsystems (carpeta para subsistemas. Ignorar)
 - ii. templates
 1. webscripts (carpeta para los descriptores y templates de los webscripts)
 - b. module
 - i. truettaAlfresco
 1. context (carpeta con los ficheros de Spring a cargar)
 2. messages (carpeta con los mensajes)
 3. model (carpeta con los modelos)
 4. alfresco-global.properties (Fichero de configuración por defecto del módulo)

5. log4j.properties (Fichero de configuración de logs)
6. module-context.xml (Fichero inicial de carga de Spring)
7. module.properties (Fichero que define el modulo)
- c. subsystems (carpeta para subsistemas. Ignorar)
2. META-INF (Ignorar)
 - a. Resources (Ignorar)
3. rebel.xml (configuración de JRebel. Ignorar)

Al iniciar el contexto de Alfresco el sistema lee el fichero module-context.xml. Este fichero incluye en el contexto los ficheros de la carpeta context.

```
<beans>
  <!-- This is filtered by Maven at build time, so that module name is single sourced. -->
  <!-- Note. The bootstrap-context.xml file has to be loaded first.
        Otherwise your custom models are not yet loaded when your service beans are instantiated and you
        cannot for example register policies on them. -->
  <import resource="classpath:alfresco/module/${project.artifactId}/context/bootstrap-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/service-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/webscript-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/action-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/bonita-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/filter-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/behaviour-context.xml" />
</beans>
```

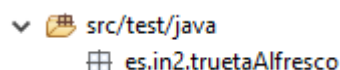


Dentro de estos ficheros estarán todos los beans del contexto que añadimos. Por ejemplo:

```
<bean id="iniciar-workflow-action" class="es.in2.truetAlfresco.actions.IniciarWorkflow"
      parent="action-executer">
  <property name="bonitaService" ref="bonitaServiceImpl" />
</bean>
```

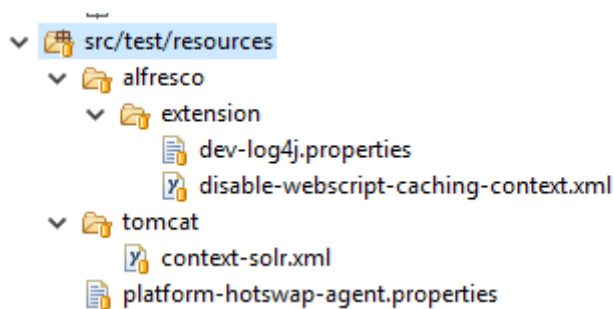
1.1.3 src/test/java

Carpeta con las clases para testing.



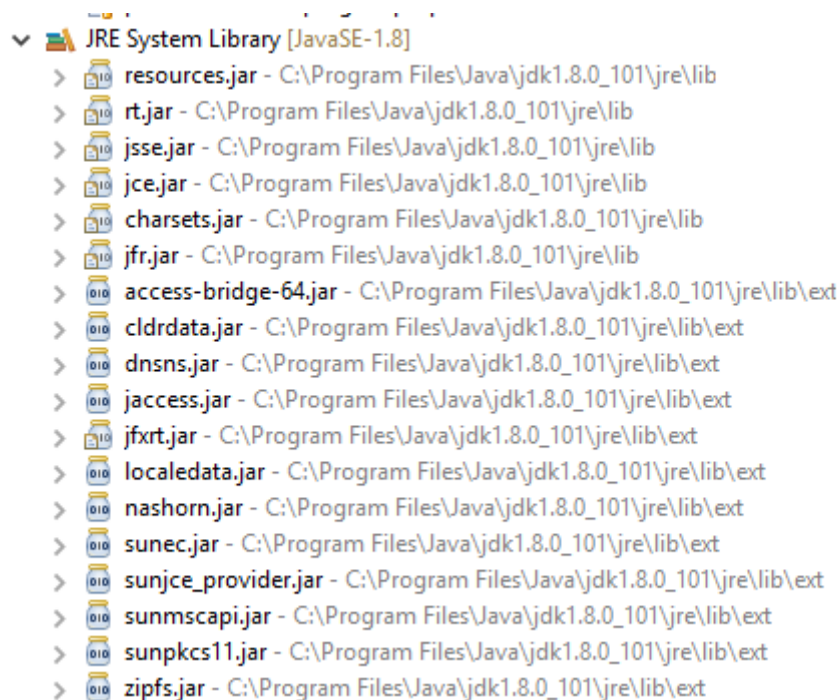
1.1.4 src/test/resources

Carpeta con recursos para testing



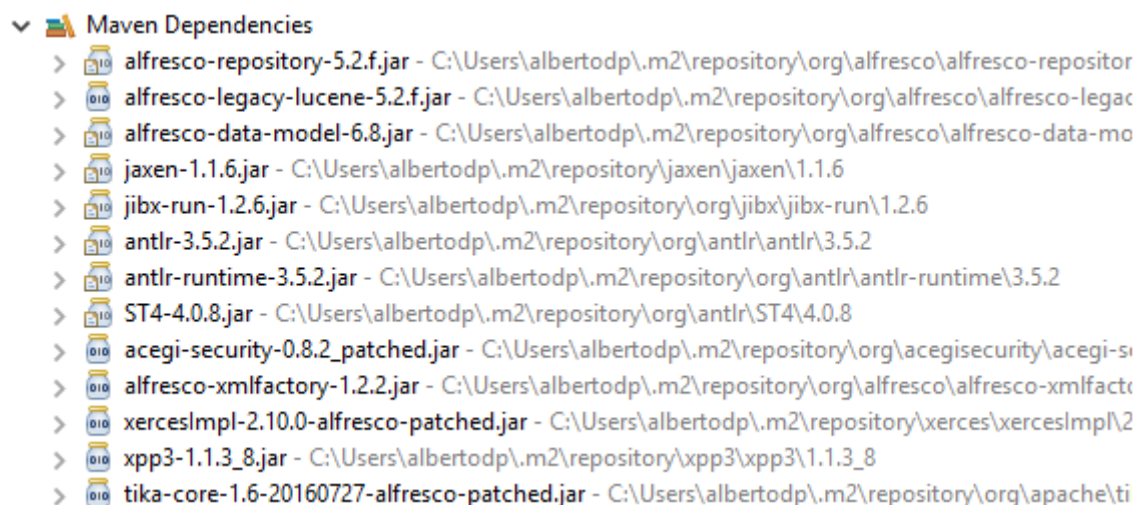
1.1.5 JRE System Library

Carpeta “virtual” que nos muestra las librerías de la maquina virtual



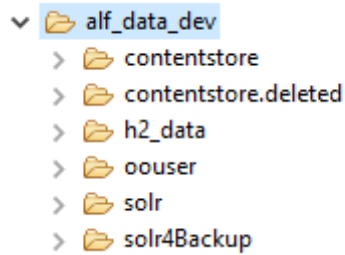
1.1.6 Maven Dependencies

Carpeta “virtual” que nos muestra las dependencias de maven



1.1.7 alf_data_dev

Carpeta para los contenidos, base de datos e indices



1. contentstore (contenidos subidos)
2. contentstore.deleted (contenidos borrados de la papelera)
3. h2_data (base de datos)
4. oouser (carpeta de trabajo para Open Office)
5. solr (carpeta para los índices)
6. solr4Backup (carpeta de backup para los índices)

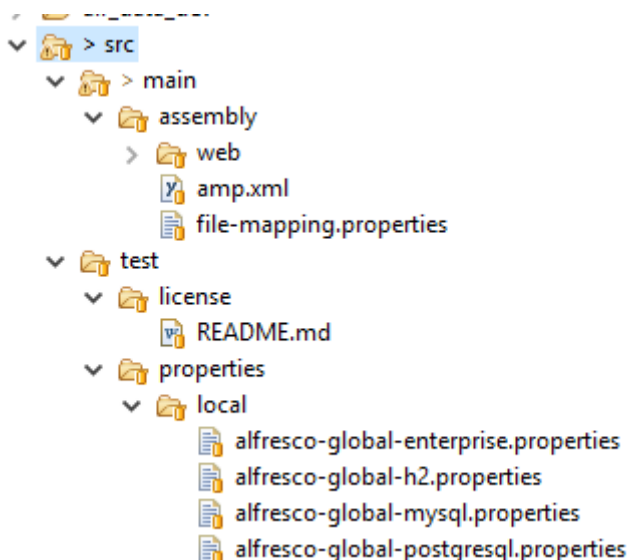
1.1.8 src

En la carpeta src tendremos los ficheros de configuración para la generación del AMP y los ficheros alfresco-global*.properties.

Los ficheros de assembly los dejaremos como están.

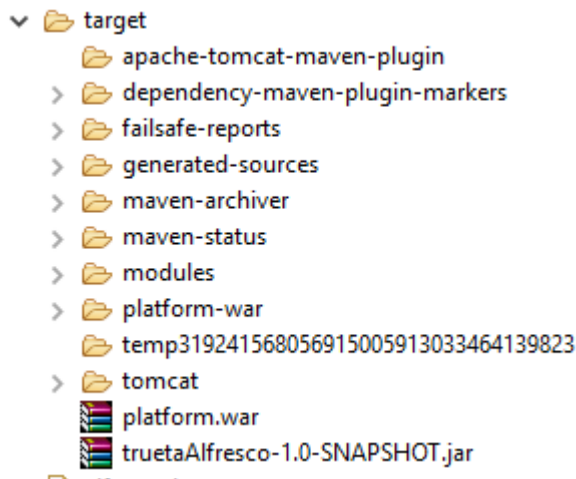
De los ficheros de configuración de alfresco nos interesará el alfresco-global-h2.properties.

En este fichero pondríamos parámetros de configuración de alfresco que afectan solo al alfresco de desarrollo. Estos parámetros los deberemos configurar en el alfresco-global.properties del alfresco de producción.

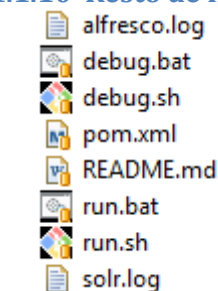


1.1.9 target

En la carpeta target se almacenan los compilados. De esta carpeta nos interesa el "truetaAlfresco-1.0-SNAPSHOT.jar". Este será el fichero que subiremos al alfresco de producción.



1.1.10 Resto de ficheros



De los siguientes ficheros ignoraremos los *.bat y *.sh. Estos ficheros están para arrancar desde consola el alfresco de desarrollo.

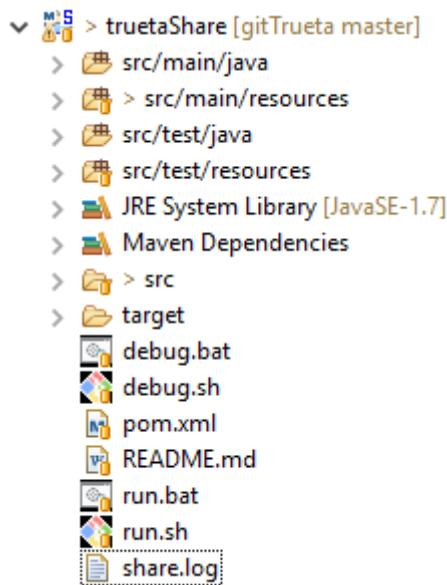
Los ficheros de log también los podremos ignorar ya que el log nos aparece en la vista de logs del eclipse. Estos ficheros también están por si la ejecución se hace desde consola.

El fichero README.md es un estándar de documentación.

Por último, el fichero pom.xml, nos permite configurar las dependencias del proyecto.

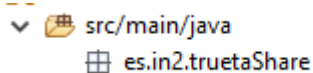
1.2 Proyecto Share

La estructura del proyecto es la siguiente:



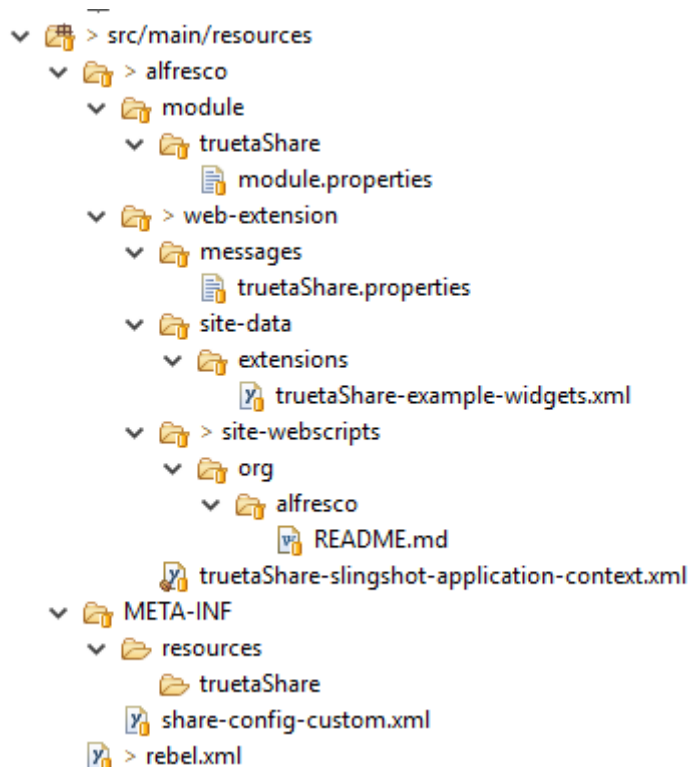
1.2.1 src/main/java

En la carpeta src/main/java tendremos los fuentes.



1.2.2 src/main/resources

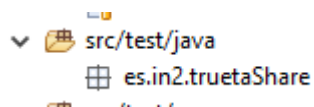
En la carpeta src/main/resources tendremos los ficheros de recursos, es decir, los ficheros que los fuentes necesitan.



- alfresco
 - a. module
 - i. truetaShare
 - 1. module.properties (fichero de configuración del módulo)
 - b. web-extension
 - i. messages (ficheros con los textos para i18n)
 - ii. site-data (carpeta para añadir paginas/widgets a Share)
 - iii. site-webscripts (carpeta para webScripts de share. API rest con html como respuesta)
- META-INF
 - a. Resources
 - i. truetaShare
 - b. share-config-custom.xml (fichero para configurar apariencia del Share. Este fichero nos servirá para configurar formularios para hacer pruebas y revisar meta-datos)

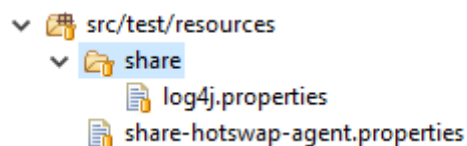
1.2.3 src/test/java

Carpeta para testing



1.2.4 src/test/resources

Carpeta para los recursos de testing



1.2.5 JRE System Library

Carpeta “virtual” que nos muestra las librerías de la maquina virtual

- ▼ JRE System Library [JavaSE-1.8]
 - > resources.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib
 - > rt.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib
 - > jsse.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib
 - > jce.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib
 - > charsets.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib
 - > jfr.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib
 - > access-bridge-64.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > cldrdata.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > dnsns.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > jaccess.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > jfxrt.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > localedata.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > nashorn.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > sunec.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > sunjce_provider.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > sunmscapi.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > sunpkcs11.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext
 - > zipfs.jar - C:\Program Files\Java\jdk1.8.0_101\jre\lib\ext

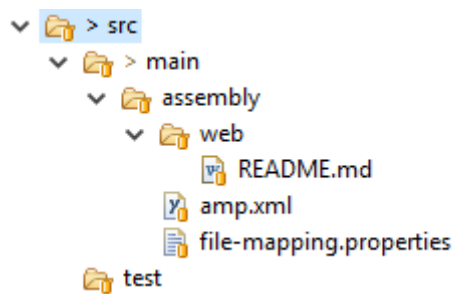
1.2.6 Maven dependencies

Carpeta “virtual” que nos muestra las dependencias de Maven

- ▼ Maven Dependencies
 - > share-5.2.d-classes.jar - C:\Users\albertodp\.m2\repository\org\alfresco\share\5.2.d
 - > alfresco-web-framework-commons-5.2.d-classes.jar - C:\Users\albertodp\.m2\repository\org\alfresco\share\5.2.d-classes.jar
 - > alfresco-core-6.5.jar - C:\Users\albertodp\.m2\repository\org\alfresco\alfresco-core\6.5
 - > commons-codec-1.10.jar - C:\Users\albertodp\.m2\repository\commons-codec\commons-codec\1.10
 - > commons-collections-3.2.2.jar - C:\Users\albertodp\.m2\repository\commons-collections\commons-collections\3.2.2
 - > commons-httpclient-3.1-HTTPCLIENT-1265.jar - C:\Users\albertodp\.m2\repository\commons-httpclient\commons-httpclient\3.1-HTTPCLIENT-1265
 - > commons-logging-1.2.jar - C:\Users\albertodp\.m2\repository\commons-logging\commons-logging\1.2
 - > commons-io-2.4.jar - C:\Users\albertodp\.m2\repository\commons-io\commons-io\2.4
 - > commons-math3-3.6.1.jar - C:\Users\albertodp\.m2\repository\org\apache\commons\commons-math3\3.6.1
 - > jug-2.0.0-asl.jar - C:\Users\albertodp\.m2\repository\org\safehaus\jug\jug\2.0.0
 - > log4j-1.2.17.jar - C:\Users\albertodp\.m2\repository\log4j\log4j\1.2.17
 - > json-20160212.jar - C:\Users\albertodp\.m2\repository\org\json\json\20160212
 - > spring-orm-3.2.16.RELEASE.jar - C:\Users\albertodp\.m2\repository\org\springframework\spring-orm\3.2.16.RELEASE
 - > spring-jdbc-3.2.16.RELEASE.jar - C:\Users\albertodp\.m2\repository\org\springframework\spring-jdbc\3.2.16.RELEASE
 - > spring-tx-3.2.16.RELEASE.jar - C:\Users\albertodp\.m2\repository\org\springframework\spring-tx\3.2.16.RELEASE
 - > spring-context-3.2.16.RELEASE.jar - C:\Users\albertodp\.m2\repository\org\springframework\spring-context\3.2.16.RELEASE
 - > spring-aop-3.2.16.RELEASE.jar - C:\Users\albertodp\.m2\repository\org\springframework\spring-aop\3.2.16.RELEASE
 - > aopalliance-1.0.jar - C:\Users\albertodp\.m2\repository\org\springframework\spring-aop\1.0
 - > spring-expression-3.2.16.RELEASE.jar - C:\Users\albertodp\.m2\repository\org\springframework\spring-expression\3.2.16.RELEASE
 - > spring-context-support-3.2.16.RELEASE.jar - C:\Users\albertodp\.m2\repository\org\springframework\spring-context-support\3.2.16.RELEASE
 - > jaxb-xjc-2.2.7.jar - C:\Users\albertodp\.m2\repository\com\sun\xml\bind\jaxb-xjc\2.2.7
 - > jaxb-core-2.2.7.jar - C:\Users\albertodp\.m2\repository\com\sun\xml\bind\jaxb-core\2.2.7
 - > jaxb-api-2.2.7.jar - C:\Users\albertodp\.m2\repository\com\sun\xml\bind\jaxb-api\2.2.7
 - > istack-commons-runtime-2.16.jar - C:\Users\albertodp\.m2\repository\com\sun\istack\istack-commons-runtime\2.16

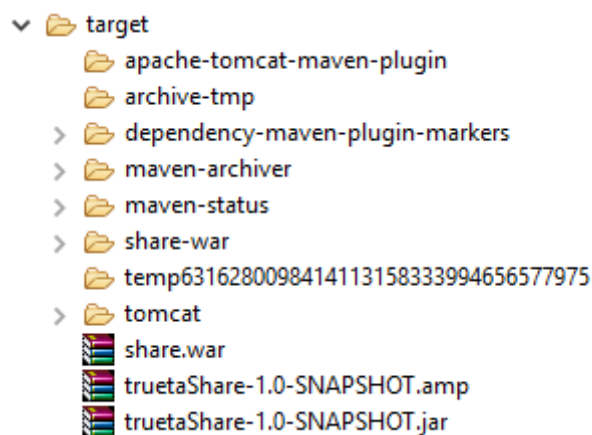
1.2.7 src

En la carpeta src tendremos los ficheros de configuración para la generación del AMP.

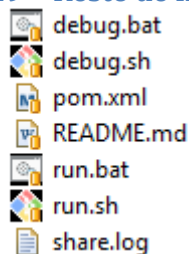


1.2.8 target

En la carpeta target se almacenan los compilados. De esta carpeta nos interesa el “truetaShare-1.0-SNAPSHOT.jar”. Este será el fichero que subiremos al alfresco de producción.



1.2.9 Resto de ficheros



De los siguientes ficheros ignoraremos los *.bat y *.sh. Estos ficheros están para arrancar desde consola el alfresco de desarrollo.

Los ficheros de log también los podremos ignorar ya que el log nos aparece en la vista de logs del eclipse. Estos ficheros también están por si la ejecución se hace desde consola.

El fichero README.md es un estándar de documentación.

Por último, el fichero pom.xml, nos permite configurar las dependencias del proyecto.

2 Extensiones

2.1 Alfresco

2.1.1 Modelo

Para crear un modelo (conjunto de tipos) hemos de añadir un fichero en la carpeta “src/main/resources/alfresco/module/truetaAlfresco”.

Este fichero es un xml en el que definiremos los diferentes tipos. Hemos de tener en cuenta que para que alfresco cargue estos modelos hemos de cargarlo en el fichero “src/main/resources/alfresco/module/truetaAlfresco/context/bootstrap-context.xml”.

```
<!-- Registration of new models -->
<bean id="truetaAlfresco.dictionaryBootstrap" parent="dictionaryModelBootstrap"
      depends-on="dictionaryBootstrap">
  <property name="models">
    <list>
      <value>alfresco/module/${project.artifactId}/model/trueta-model.xml</value>
    </list>
  </property>
  <property name="Labels">
    <list>
      <value>alfresco/module/${project.artifactId}/messages/trueta</value>
    </list>
  </property>
</bean>
```

Como vemos en la imagen estamos cargando un bean que hereda de “dictionaryModelBootstrap” y depende de “dictionaryBootstrap”.

La dependencia en este caso sirve para que no se cargue nuestro bean con nuestros models hasta que no se hayan cargado los modelos de Alfresco. Es muy importante porque nuestros modelos importan y heredan los modelos de alfresco.

En el apartado “Labels” definimos el fichero properties con las etiquetas de los formularios. Este sistema nos permite crear un fichero por idioma.

Hemos de tener en cuenta que el fichero “bootstrap-context.xml” se carga en el arranque porque está definido en el fichero “module-context.xml”.

```
<beans>
  <!-- This is filtered by Maven at build time, so that module name is single sourced. -->
  <!-- Note. The bootstrap-context.xml file has to be loaded first.
       Otherwise your custom models are not yet loaded when your service beans are instantiated and you
       cannot for example register policies on them -->
  <import resource="classpath:alfresco/module/${project.artifactId}/context/bootstrap-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/service-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/webscript-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/action-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/bonita-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/filter-context.xml" />
  <import resource="classpath:alfresco/module/${project.artifactId}/context/behaviour-context.xml" />
</beans>
```

Es muy importante que la carga de los modelos sea lo primero ya que el resto de beans (behaviours, actions, webscripts, etc...) utilizaran los modelos.

2.1.2 Action

Para crear una acción hemos de crear una clase que extienda de “ActionExecuterAbstractBase”

```
public class GeneratePdfFromTemplateAction extends ActionExecuterAbstractBase {
```

Una vez creada la clase nos obligará a extender el método “executeImpl”

```
@Override  
protected void executeImpl(Action action, NodeRef node) {
```

Será en este método donde pondremos la lógica de nuestra acción. El método recibe 2 parámetros:

1. Action
 - a. La variable acción no sirve para recoger los parámetros si los hubiera
2. Node
 - a. Es el nodo de alfresco sobre el que se ejecuta la acción.

```
String tipo = (String) action.getParameterValue("tipo");
```

Para inyectar servicios a nuestra acción hemos de definirlos como atributos de la clase y crear los métodos setter/getter de estos atributos. Estos métodos son vitales para que Spring pueda inyectar.

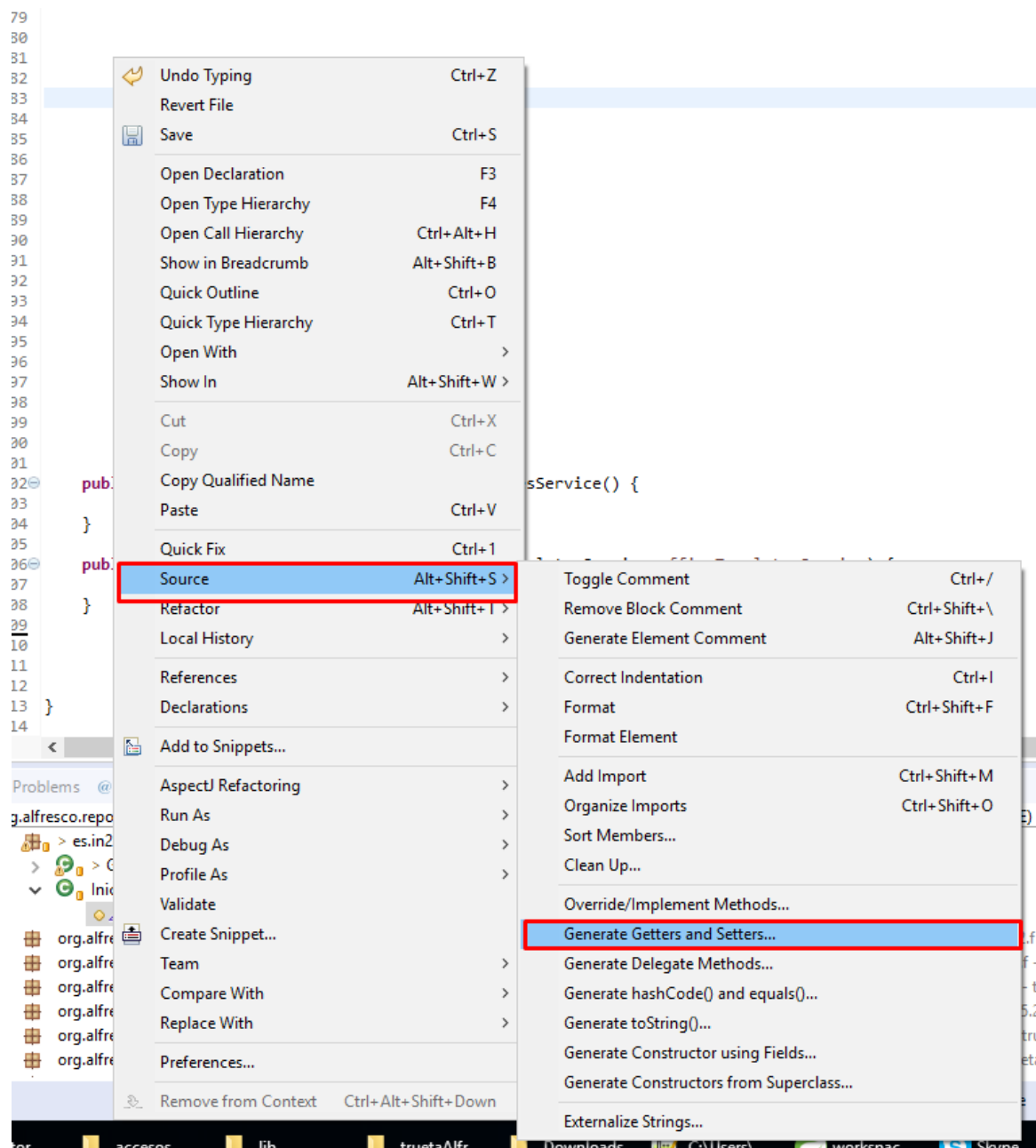
Atributos

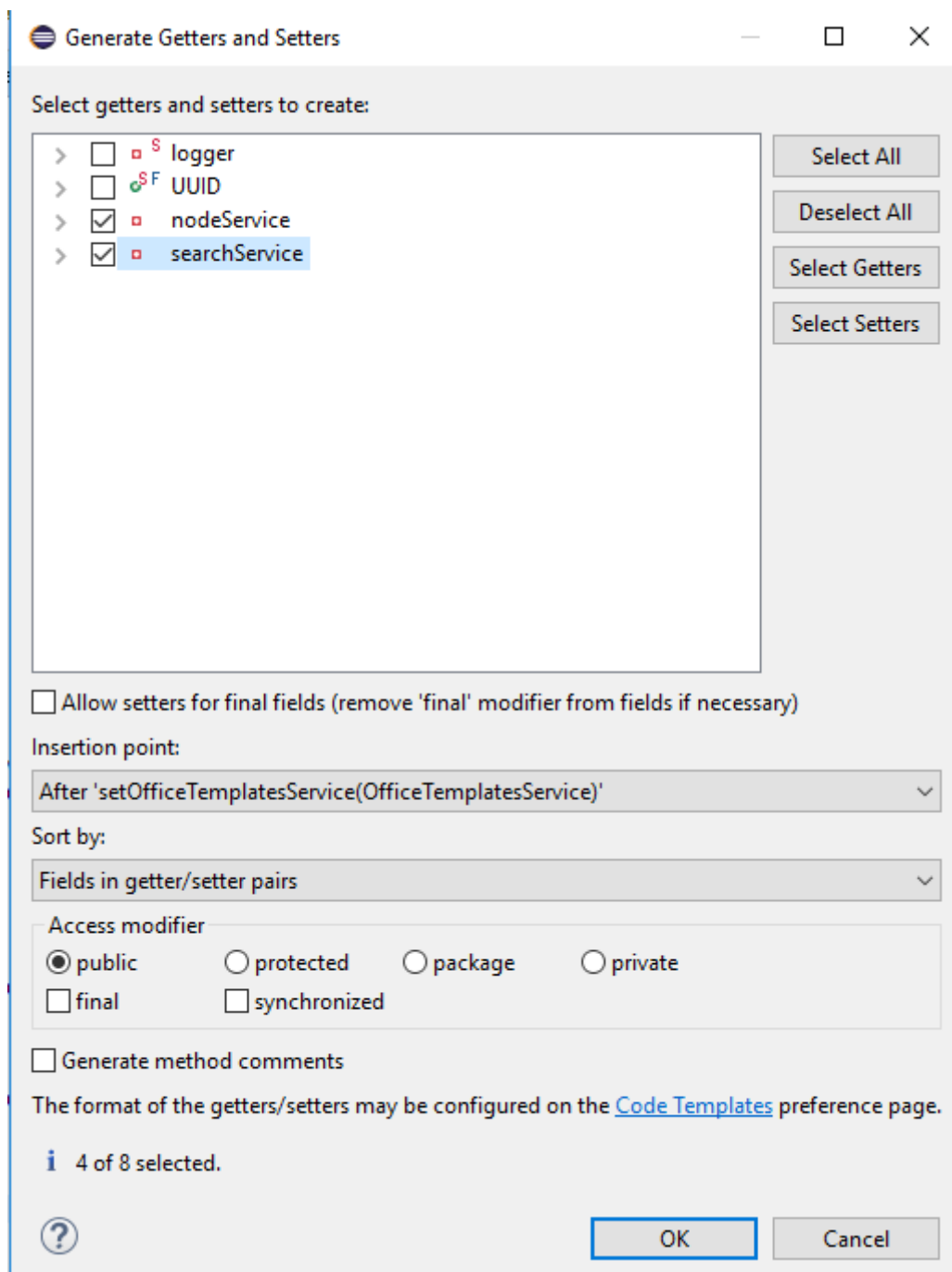
```
private NodeService nodeService;  
private SearchService searchService;
```

Setters/Getters

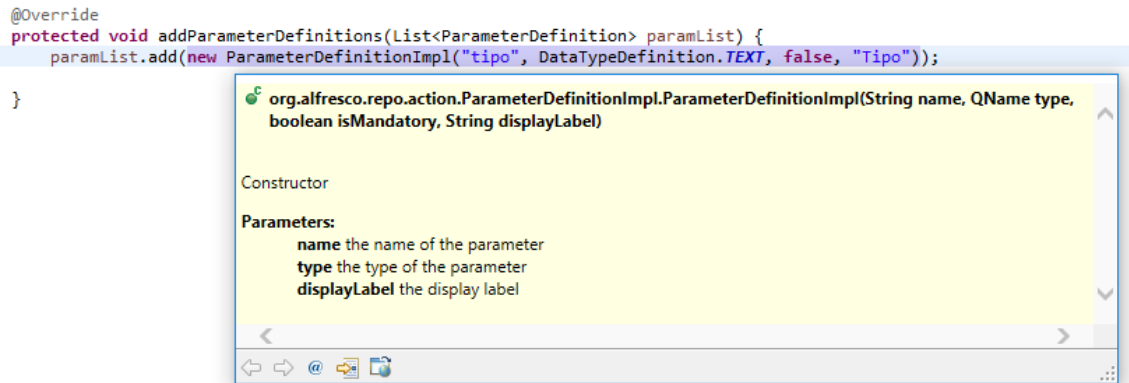
```
public NodeService getNodeService() {  
    return nodeService;  
}  
  
public void setNodeService(NodeService nodeService) {  
    this.nodeService = nodeService;  
}  
  
public SearchService getSearchService() {  
    return searchService;  
}  
  
public void setSearchService(SearchService searchService) {  
    this.searchService = searchService;  
}
```

Para generar los métodos podemos utilizar una funcionalidad de eclipse y nos aseguramos que los métodos tienen la firma correcta.





Si quisiéramos definir parámetros en la acción existe un método para ello.



Una vez implementada la acción hemos de inyectar el bean en el contexto de Alfresco mediante el fichero “src/main/resources/alfresco/module/truetaAlfresco/context/action-context.xml”

```
<bean id="generatePdfFromTemplateAction" class="es.in2.truetaAlfresco.actions.GeneratePdfFromTemplateAction"
    parent="action-executer">
    <property name="officeTemplatesService" ref="officeTemplatesService" />
    <property name="nodeService" ref="nodeService" />
    <property name="searchService" ref="searchService" />
</bean>
```

Para cada acción tendremos un bean con la siguiente configuración.

- id: El identificador ha de ser único y definirá el nombre para invocar la acción
- class: La clase que hemos implementado
- parent: Siempre ha de ser action-executer
- bloque de property: Para cada servicio que queramos inyectar hemos de añadir una property
 - a. name: es el nombre del atributo de la clase
 - b. ref: identificador del bean






























































































Para que alfresco cargue el fichero de acciones hemos de tenerlo en el fichero “src/main/resources/alfresco/module/truetaAlfresco/module-context.xml”

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <!-- This is filtered by Maven at build time, so that module name is single sourced. -->
    <!-- Note. The bootstrap-context.xml file has to be loaded first.
         Otherwise your custom models are not yet loaded when your service beans are instantiated and you
         cannot for example register policies on them. -->
    <import resource="classpath:alfresco/module/${project.artifactId}/context/bootstrap-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/service-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/webscript-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/action-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/bonita-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/filter-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/behaviour-context.xml" />
</beans>
```

2.1.3 Behaviour

Para crear un behaviour hemos de crear una clase que implemente de una de las siguientes interfaces.

- ▼  ClassPolicy
 - >   AfterCreateVersionPolicy
 - >   AfterVersionRevertPolicy
 - >   BeforeAddAspectPolicy
 - >   BeforeArchiveNodePolicy
 -   BeforeCancelCheckOut
 -   BeforeCheckIn
 - >   BeforeCheckOut
 - >   BeforeCopyPolicy
 - >   BeforeCreateNodePolicy
 -   BeforeCreateStorePolicy
 -   BeforeCreateVersionPolicy
 - >   BeforeDeleteNodePolicy
 - >   BeforeLock
 - >   BeforeMoveNodePolicy
 - >   BeforePurgeNodePolicy
 - >   BeforeRemoveAspectPolicy
 -   BeforeSetNodeTypePolicy
 -   BeforeStartInboundTransferPolicy
 - >   BeforeUpdateNodePolicy
 - >   CalculateVersionLabelPolicy
 - >   OnAddAspectPolicy
 -   OnAddFavouritePolicy
 -   OnAsyncActionExecute
 - >   OnCancelCheckOut
 - >   OnCheckIn
 - >   OnCheckOut
 -   OnContentPropertyUpdatePolicy
 - >   OnContentReadPolicy
 - >   OnContentUpdatePolicy
 - >   OnCopyCompletePolicy
 - >   OnCopyNodePolicy
 - >   OnCreateNodePolicy
 -   OnCreateStorePolicy
 - >   OnCreateVersionPolicy
 - >   OnDeleteNodePolicy
 -   OnEndInboundTransferPolicy
 -   OnLoadDynamicModel
 - >   OnMoveNodePolicy
 - >   OnRemoveAspectPolicy
 -   OnRemoveFavouritePolicy
 - >   OnRestoreNodePolicy
 -   OnRevertVersionPolicy
 -   OnSetNodeTypePolicy
 -   OnStartInboundTransferPolicy
 - >   OnUpdateNodePolicy
 - >   OnUpdatePropertiesPolicy

```
public class CustomBehaviour implements OnCreateNodePolicy{
```

```

private PolicyComponent policyComponent;
private Behaviour onCreateNode;

private ActionService actionService;
private NodeService nodeService;

public void init() {
    this.onCreateNode = new JavaBehaviour(this, "onCreateNode", Behaviour.NotificationFrequency.TRANSACTION_COMMIT);
    this.policyComponent.bindClassBehaviour(NodeServicePolicies.OnCreateNodePolicy.QNAME,
        ContentModel.TYPE_CONTENT, this.onCreateNode);
}

```

```

@Override
public void onCreateNode(ChildAssociationRef childAssocRef) {
    NodeRef node = childAssocRef.getChildRef();
    String name = (String) nodeService.getProperty(node, ContentModel.PROP_NAME);

    if(name.equals("test.docx")){
        Action action = actionService.createAction("generatePdfFromTemplateAction");
        actionService.executeAction(action, node);
    }
}

```

```

<bean id="customBehaviour" class="es.in2.truetaAlfresco.behaviours.CustomBehaviour" init-method="init">
    <property name="policyComponent" ref="policyComponent" />
    <property name="actionService" ref="actionService" />
    <property name="nodeService" ref="nodeService" />
</bean>

```

```

<beans>
    <!-- This is filtered by Maven at build time, so that module name is single sourced. -->
    <!-- Note. The bootstrap-context.xml file has to be loaded first.
        Otherwise your custom models are not yet loaded when your service beans are instantiated and you
        cannot for example register policies on them. -->
    <import resource="classpath:alfresco/module/${project.artifactId}/context/bootstrap-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/service-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/webscript-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/action-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/bonita-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/filter-context.xml" />
    <import resource="classpath:alfresco/module/${project.artifactId}/context/behaviour-context.xml" />
</beans>

```

2.1.4 WebScript

Para crear un Webscript tenemos que crear una clase que extienda de “DeclarativeWebScript”.

Al extender de esta clase nos obliga a implementar el método “executeImpl”.

```

public class ContadorWebScript extends DeclarativeWebScript {

    private ContadorService contadorService;

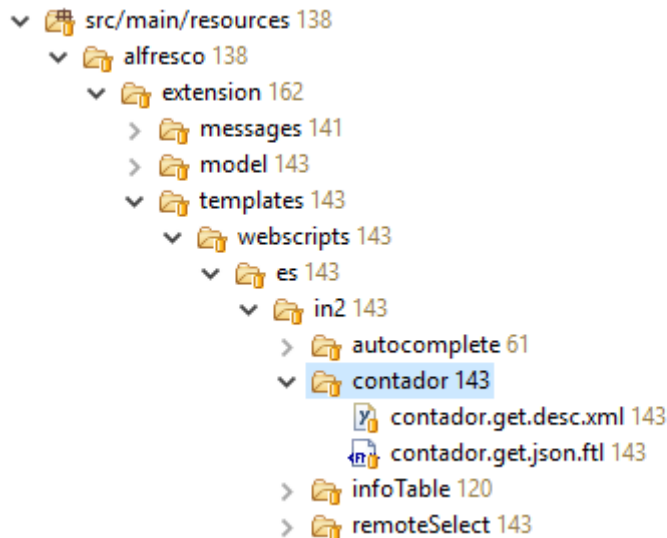
    protected Map<String, Object> executeImpl(WebScriptRequest req, Status status, Cache cache) {
        Map<String, Object> model = new HashMap<String, Object>();
        model.put("contador", contadorService.getNext(req.getParameter("id")));
        return model;
    }
}

```

El método executeImpl recibe 3 parametros y retorna un Map.

- WebScriptRequest: Objeto con los parámetros de la petición REST. Sirve para recoger los parámetros
- Status: Objeto para código HTTP de respuesta
- Cache: Objeto para gestionar cache de las peticiones
- Modelo de retorno: El modelo de retorno lo utilizamos para guardar los valores de respuesta que formatearemos en la plantilla FTL.

Los webscripts han de estar dentro de la carpeta “extensión/templates/webscripts”. A partir de esta ruta podremos crear la estructura que deseemos. Para el ejemplo: “es/in2/contador”. Esta ruta la tendremos que tener en cuenta para la inyección del webscript



Un webScript se compone de 2 ficheros de definición

- name.method.desc.XML
- name.method.json.ftl

El fichero descriptor indica el endpoint que estamos definiendo. En el ejemplo será:

<http://localhost:8080/alfresco/service/contador?id=1>. El parámetro id lo recogeremos en el código de la clase.

```

1 <webscript>
2   <shortname>Contador Webscript</shortname>
3   <description>Contador Webscript</description>
4   <url>/contador?id={id}</url>
5   <authentication>admin</authentication>
6   <format default="json"></format>
7 </webscript>
  
```

La plantilla FLT (Freemarker) nos permite componer la respuesta en base al modelo de retorno de la clase java.

```

model.put("contador", contadorService.getNext(req.getParameter("id")));
  
```

```

1 {
2   "value" : ${contador}
3 }

```

Al inyectar el bean el campo id se compone de la siguiente manera:

Webscript.(pathFicheroDescriptor).nombre.method



```

<bean id="webscript.es.in2.contador.contador.get" class="es.in2.alfrescoUtils.contador.webscript.ContadorWebScript">
  <property name="contadorService" ref="contadorService" />
</bean>

```

2.2 Share

2.2.1 Formularios

Los formularios se definen en el fichero share-config-custom.xml

```

<config evaluator="model-type" condition="phcd:impresoraList">
  <forms>
    <form>
      <field-visibility>
        <show id="phcd:nombreImpresora" />
        <show id="phcd:descripcionImpresora" />
      </field-visibility>
    </form>
  </forms>
</config>

<config evaluator="node-type" condition="phcd:impresoraList">
  <forms>
    <form>
      <field-visibility>
        <show id="phcd:nombreImpresora" />
        <show id="phcd:descripcionImpresora" />
      </field-visibility>
    </form>
  </forms>
</config>

```

El bloque model-type es para los formularios de creación y el bloque node-type es para la edición y visualización.