Business logic addon

Business logic addon

- Introduction
- Addon spring descriptor
- Addon startup
- Authorizations descriptor
- Data schema descriptor

Introduction

A business logic addon should follow the conventions and guidelines used to develop Soffid IAM.

It must be composed of:

1. Hibernate entities and mappings.

2. DAOs to access those hibernate entities. They must contain transformation methods to build value objects from entities and vice versa.

Service objects to manage value objects.

3. EJBs that wrap access to service objects. They are responsible for security access control. They must be Stateless Session Beans with Container transaction management.

4. Authorizations descriptor.

5. Spring descriptor.

Addon spring descriptor

Addons must have an addon-applicationProperties.xml spring descriptor.

Hibernate DAOS should be declared as the next example states. They can reference any bean present at the Soffid console, including sessionFactory. It's advisable to use two global interceptors:

- **hibernateInterceptor** creates and close the hibernate session when needed.
- **daoInterceptor-*** is a placeholder for future DAO interceptors.

```
<!-- ========================== HIBERNATE DAOs ========================== -->
<!-- SampleEntity DAO -->
<bean id="sampleEntityDao" class="org.springframework.aop.framework.ProxyFactoryBean">
   <property name="target">
      <bean class="com.soffid.iam.addons.sample.model.SampleEntityDaoImpl">
         <property name="sessionFactory"><ref bean="sessionFactory"/></property>
      </bean>
   </property>
   <property name="proxyInterfaces">
      <value>com.soffid.iam.addons.sample.model.SampleEntityDao</value>
   </property>
   <property name="interceptorNames">
      <list>
         <value>hibernateInterceptor</value>
         <value>daoInterceptor-*</value>
      </list>
   </property>
</bean>
```

Spring services should be declared as the next example states. They can reference any bean present at Soffid console. It's advisable to use the global interceptors:

- **transactionInterceptor**: starts and finishes the user transaction when needed.
- **hibernateInterceptor** creates and closes the hibernate session when needed.
- **serviceInterceptor-*** is a placeholder for future service interceptors.

```
<!-- sampleService Service Proxy with inner sampleService Service Implementation -->
<bean id="sampleService" class="org.springframework.aop.framework.ProxyFactoryBean">
```

```
    <property name="target">
        <bean class="com.soffid.iam.addons.sample.service.SampleServiceImpl">
        </bean>
    </property>
    <property name="proxyInterfaces">
        <value>com.soffid.iam.addons.sample.service.ReplicaBootService</value>
    </property>
    <property name="interceptorNames">
        <list>
            <value>serviceTransactionInterceptor</value>
            <value>hibernateInterceptor</value>
            <value>serviceInterceptor-*</value>
        </list>
    </property>
</bean>
```

Finally, the hibernate mapping resources should be listed at a special bean, just like the following example:

```
<!-- ========================== HIBERNATE PROPERTIES
========================== -->
<bean id="hibernate-addon-replica" class="es.caib.seycon.ng.spring.AddonHibernateBean">
    <property name="mappingResources">
    <list>
        <value>com/soffid/iam/addons/sample/model/SampleEntity.hbm.xml</value>
        </list>
</property>
</bean>
```

Addon startup

Some addons need to execute some code on startup. To achieve this, the startup must implement and declare a SpringService that implements the es.caib.seycon.ng.servei.ApplicationBootService interface.

This interface has two methods that will be called after synchronization server or console startup :

```
public interface ApplicationBootService {
    /**
     * Operation syncServerBoot - synchronization server engine boot
     */
    void syncServerBoot()
            throws es.caib.seycon.ng.exception.InternalErrorException;
    /**
     * Operation consoleBoot - console boot
     */
    void consoleBoot()
            throws es.caib.seycon.ng.exception.InternalErrorException;
}
```

Authorizations descriptor

The authorization descriptor is an XML describing the different authorizations that can be granted to users.

This file should be located at /com/soffid/addon/authorization.xml

The template for this file is:

```
<?xml version="1.0" encoding="utf-8"?>
<autoritzacions>
 <autoritzacio>
            <!-- Required: authorization name -->
      <codi>addon:privacy:create</codi>
            <!-- Required: Functional area description -->
      <ambit>Privacy</codi>
            <!-- Required: authorization full description -->
      <descripcio>Create privacy records</descripcio>

            <!-- Optional: scope of the authorization. It can contain a comma separated list of -->
            <!-- GRUP => Group scoped role -->
            <!-- APLICACIONS => Information system scoped role -->
      <tipusDomini>GRUPS</tipusDomini>
            <!-- Optional: how the authorization will spread along business units -->
            <!-- children => the permission will be effective on business unit and its children -->
            <!-- parent   => the permission will be effective on business unit and its parents -->
            <!-- both     => the permission will be effective on business units, its parents and its children -->
      <scope>children</scope>
            <!-- Permissions that will be granted to anyone with this permission -->
      <hereta>
         user:query,
         group:query
      </hereta>
   </autoritzacio>
   ...
</autortizacions>
```

Data schema descriptor

Schema description

The schema must be expressed as an XML file. This file should be located at a core addon module and be named plugin-ddl.xml.

This DDL file can contain descriptions for tables, indexes and foreign keys.

Tables

A table is composed of a <table> entity containing one or more <column> entities:

Attributes for <table> entity	
name	Table name

Attributes for <column> entity	
name	Column name
type	It can have one of the following values, as stated at java.sql.Types class: INTEGER LONG BOOLEAN / BIT FLOAT DOUBLE VARCHAR CHAR DATE CLOB BLOB
length	Optional column size.
notNull	**true** if it cannot contain null values
primaryKey	**true** if it's part of the primary key. The primary key will be composed of all attributes marked as primary key. The order of them will be the same as their appear order at XML fi

Foreign keys

A foreign key is declared using the <foreignKey> entity. It will contain one or more <column> entities. Each columns identifies a member of the foreign key. It will also contain one or more <foreignColumn> entities. Each foreignColumn entity specifies the name of the corresponding column at the foreign table.

Attributes for entity <foreignKey>	
name	Foreign key name
tableName	Table name
foreignTableName	Name of master table

Attributes for <column> entity	
name	Column containing the foreign key

Attributes for <foreignColumn> entity	
name	Column containing the foreign key

Indexes

An index is declared using the <index> entity. It will contain one or more <column> entities. Each column identifies a member of the index.

Attributes for <index> entity	
name	Index name
tableName	Table name
unique	**true** if no duplicated keys will be allowed

Attributes for entity <column>	
name	Column to be indexed

Sample plugin-ddl.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<database>
    <table name="t1">
        <column name="id" type="LONG" notNull="true"
            autoIncrement="true" primaryKey="true"/>
        <column name="vc" type="VARCHAR" length="55"/>
        <column name="t1date" type="DATETIME" notNull="true"/>
        <column name="userid" type="LONG" />
    </table>
    <index name="i1" table="t1">
        <column name="vc"/>
    </index>

    <table name="t2">
        <column name="id" type="LONG" primaryKey="true"/>
    </table>
    <foreignKey name="t1_fk" table="t1" foreignTable="t2">
        <column name="userid"/>
        <foreignColumn name="id"/>
    </foreignKey>
</database>
```