MANUAL API ROLSAC V2

DESEMBRE DE 2014

FULL DE CONTROL DE VERSIONS

DOCUMENT/ARXIU

Títol: **MANUAL API ROLSAC V2**	Nombre Arxiu/s: **ROLSAC-1.2-API_V2_MAN_DVLP.doc**
Codi: **ROLSAC-1.2-API_V2_MAN_DVLP**	Suport lògic: **Word**
Data: **03-12-2014**	Ubicació física:
Versió: **1**	

REGISTRE DE CANVIS

Versió	Pàgines	Mou del canvi
1		Creació del document.

DISTRIBUCIÓ DEL DOCUMENT

Nom	Personal
Estela Pisano	DGTIC
Salvador Gelabert	DGTIC
Bartomeu Cerdà	Brújula
Bartomeu Bennassar	Brújula
Antoni Garcia	AT4

CONTROL DEL DOCUMENT

PREPARAT	REVISAT/ APROVAT	ACEPTAT	ACEPTADO
Bartomeu Cerdà	Salvador Gelabert		

Emplenar amb el nom, la firma i la data.

Només per clients

MANUAL API ROLSAC V2

Introducció

L'API ROLSAC és un conjunt de mètodes que s'ofereix a tercers per poder interactuar de forma senzilla amb els continguts gestionats a través del Backoffice de ROLSAC.

Cal indicar que aquest conjunt només es vàlid per realitzar consultes, la única excepció és el servei d'estadístiques.

Actualment l'API es utilitzada per la SEUCAIB i pròximament per l'aplicació GUSITE.

Aquest document mostra l'estructura de l'API, així com un petit manual per poder realitzar modificacions, ja sigui per poder modificar funcionalitats com per afegir-ne de noves.

1 Definicions i patrons de l'estructura de l'API

1.1 Patrons de disseny utilitzats

- **Adapter:** S'utilitza per transformar una interfície en una altra, de manera que la classe pugui interactuar amb una interfície no compatible.
- **Strategy:** S'utilitza per poder triar entre diferents algoritmes per resoldre un problema de forma dinàmica. En el nostre cas, ens permet seleccionar si la nostra cridada a l'API es realitza a través dels EJB o mitjançant un WS. Així aconseguim una interfície única i mitjançant una petita configuració, podem triar entre una o l'altre segons les nostres necessitats.

1.2 Definició dels Criterias i DTOs

Els Criterias, que són els objectes que es passen com a paràmetres a les cridades de l'API per ser utilitzats posteriorment per Hibernate, són definits a partir d'un WSDL i generats els objectes Java amb el XDoclet.

Igual que els Criterias duen la informació necessària per poder realitzar la consulta, els DTO són els objectes retornats per l'API i contenen el resultat de la consulta. Igual que els Criterias, els DTO també es defineixen amb un WSDL.

Ruta: /moduls/api/etc/wsdl/dto/ (conté els Criterias i els DTOs)

1.3 Definició d'un WS

Actualment l'API consta de 34 WS, aquests es defineixen mitjançant WSDL, igual que els Criterias i els DTOs. Si es vol modificar o afegir algun mètode s'han de modificar aquests WSDL i un conjunt de classes. És important definir bé aquests WSDL ja que són responsables de generar les classes que comuniquen les interfícies amb els EJBs de la lògica de negoci. Més endavant es proporciona una guia ràpida per a la de l'API on s'explica pas a pas com realitzar aquests canvis.

Ruta: /moduls/api/etc/wsdl/

2 Esquema de l'estructura

En l'esquema que es proporciona a continuació es pot veure a simple vista l'estructura que segueix l'API per realitzar les cridades, així com una petita descripció.

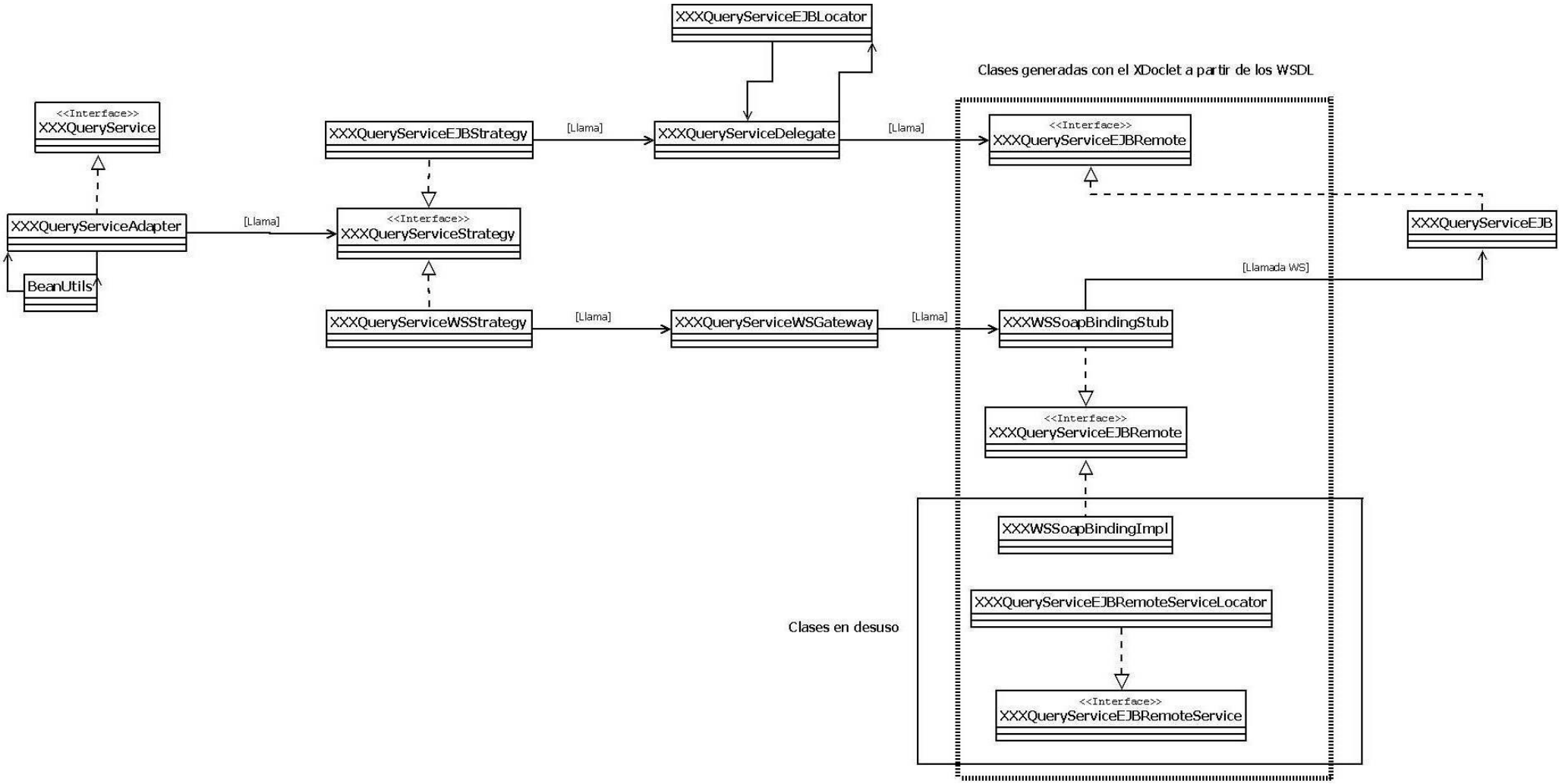
En primer lloc es crida a la capa "Adapter", la qual utilitzant la classe BeanUtils per passar a la capa "Strategy" quina estratègia utilitzarà per realitzar la cridada.

La segona capa es la "Strategy", la qual, segons la part anterior decidirà per continuar per cridar als EJBs o als WS.

Si l'opció resulta ser els EJBs, la capa "Strategy" crida al "Delegate", el qual crida directament a l'EJB on hi ha la lògica de negoci.

En canvi, si l'opció resulta ser la del WS, la capa "Strategy" crida a la capa "Gateway", la qual crida a les interfícies i classes generades per XDoclet del WSDL corresponent i així realitzar la cridada del WS.

Finalment, la lògica de negoci es troba als EJBs XXXQueryServiceEJB.java. Mitjançant aquestes capes s'aconsegueix desacoblar la lògica de la interfície de l'API.


XXXQueryServiceEJBLocator
Clases generadas con el XDoclet a partir de los WSDL
<<Interface>>
XXXQueryService
XXXQueryServiceEJBStrategy
[Llama]
XXXQueryServiceDelegate
[Llama]
<<Interface>>
XXXQueryServiceEJBRemote
XXXQueryServiceEJB
XXXQueryServiceAdapter
[Llama]
<<Interface>>
XXXQueryServiceStrategy
[Llamada WS]
BeanUtils
XXXQueryServiceWSStrategy
[Llama]
XXXQueryServiceWSGateway
[Llama]
XXXWSSoapBindingStub
<<Interface>>
XXXQueryServiceEJBRemote
XXXWSSoapBindingImpl
XXXQueryServiceEJBRemoteServiceLocator
Clases en desuso
<<Interface>>
XXXQueryServiceEJBRemoteService

2.1 Estructura bàsica de Packages de l'API

L'API s'estructura dintre del projecte en els mòduls "/moduls/api/" i "/moduls/apiws/".

En el mòdul "apiws" bàsicament hi ha l'arxiu server-config.wsdd, que és el que descriu els web services a la part del servidor.

En canvi, dintre del mòdul "api" és on hi ha tot el codi de l'API de Rolsac.

El primer directori que en trobam és "etc" on hi ha "/moduls/api/etc/wsdl" que és on s'agrupen tots els wsdl descrits en l'apartat 2.

El segon, és el directori "src" on hi ha tot el codi de l'API distribuït en 45 Packages. Quasi la totalitat dels packages són els diferents EJBs (que segueixen les capes descrites abans) interfícies i descriptor wsdd del WS sobre cada entitat. És a dir, cada package fa referència al WS que opera amb ella.

Exemple:

package es.caib.rolsac.api.v2.fetVital; fa referència al WS sobre els fets vitals i en el seu interior es subdivideix en directoris segons fan referència al WS o al EJB.

Com s'ha dit, quasi tots els packages són dels diferents WS/EJB de que disposa'm, no obstant n'hi ha uns d'especials que serveixen de base i de configuració.

Llista de packages:

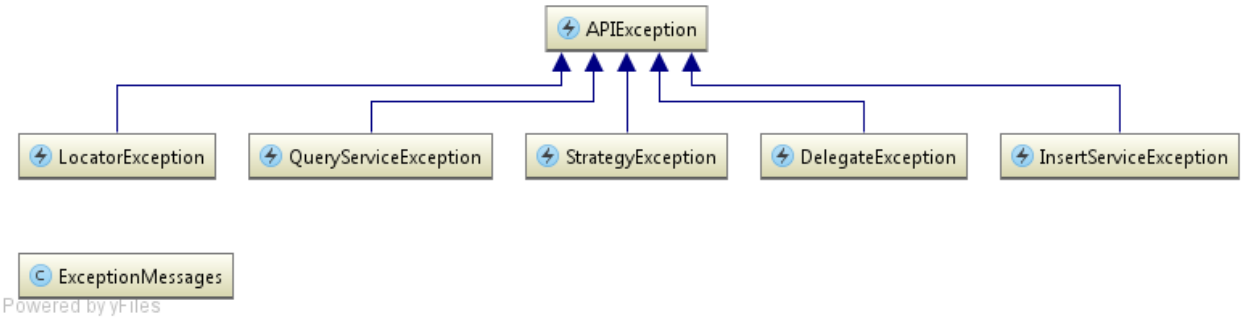
- package es.caib.rolsac.api.v2.axis.serializerFactories;
- package es.caib.rolsac.api.v2.exception;
- package es.caib.rolsac.api.v2.general;
- package es.caib.rolsac.api.v2.query;
- package es.caib.rolsac.api.v2.resources;

2.1.1 es.caib.rolsac.api.v2.serializerFactories:

Aglutina les diferents classes de configuració de l'Axis per als web services. No és necessari que tocar res d'aquest package.

2.1.2 es.caib.rolsac.api.v2.exception:

Aquest package conté totes les excepcions creades a per l'API.



APIException: Exception base de l'API.

DelegateException: Exception produïda a la capa delegate.

InsertServiceException: Exception produïda a la capa adapter.

LocatorException: Exception produïda a la capa locator.

QueryServiceException: Exception produïda a la capa adapter.

StrategyException: Exception produïda a la capa strategy.

ExceptionMessages: Són els missatges de les excepcions.

2.1.3 es.caib.rolsac.api.v2.general:

El package general aglutina totes les classes que s'empren per tots els WS. Tota classe que sigui alguna "utilitat" o classe base sobre la que estendran la resta de WS, ha d'anar dintre d'aquest package. Un exemple clar és la classe "HibernateEJB" de la qual estenen tots els EJBs de l'API on hi ha la lògica de negoci.

Classes principals:

- AxisSSLSocketFactory: Classe encarregada de recuperar el certificat del keystore i de configurar el SSLContext.
- BeanUtils: Classe encarregada de crear el "AplicationContex" i de compondre els noms dels adapters.
- CertificadoUtil: Classe que facilita la configuració de la classe AxisSSLSocketFactory, amb aquesta sols basta introduir el nom del keystore i la

seva contrasenya.

- ConfiguracioServeis: Aquesta aglutina els noms dels diferents serveis webs i un mètode que retorna la URL del servei a partir d'un nom concret. Aquest mètode recupera d'una propietat anomenada "es.caib.rolsac.api.v2.urlPrefix" la URL base sobre la qual es munten els webs services. Configuració per defecte:

es.caib.rolsac.api.v2.urlPrefix = **http://localhost:8080/sacws-api/services/**

- HibernateEJB: Bean amb la funcionalitat bàsica per interactuar amb les sessions d'Hibernate i algunes utilitats més. És la classe de la qual deriven la resta de EJBs.

2.1.4 es.caib.rolsac.api.v2.query:

Aquest package conté el conjunt de classes que s'utilitzen per la composició de les sentencies HQL. Les diferents classes serveixen per formar correctament les diferents parts de la sentencia. Aquestes utilitats només serveixen per generar sentències de consulta, ja que l'API no permet realitzar insercions, actualitzacions o eliminacions (exceptuant el servei d'estadístiques).

Com emprar:

1.- Primer es defineix un array de la classe FromClause, on cada objecte de FromClause representa una entitat de la clàusula "from" del codi HQL.

2.- Després es crea el constructor de la query al qual se li proporciona la informació bàsica per poder generar posteriorment la seqüència:

```
public QueryBuilder(String selectAlias
                  , List<FromClause> fromClauses
                  , String i18nLang
                  , String i18nAlias
                  , boolean countFlag) throws QueryBuilderException;
```

- selectAlias: És correspon amb el "select" de la sentència HQL.
- fromClauses: Array de les entitats necessàries en la sentència HQL.
- i18nLang: Idioma a recuperar, en cas que sigui una entitat que tingui traduccions.
- i18nAlias: Àlies de la traducció de l'entitat.
- countFlag: Boolean per indicar si la consulta ha de retornar un count();

3.- Afegir les restriccions de la cerca mitjançant els distints mètodes proporcionats al QueryBuilder.

4.- A continuació amb el mètode createQuery del QueryBuilder es genera la sentència.

5.- Finalment sols queda executar la query amb un dels mètodes disponibles.

Exemple:

```
public int getNumProcediments(ProcedimentCriteria procedimentCriteria) {

   Integer numResultats = -1;
   List<CriteriaObject> criteris = new ArrayList<CriteriaObject>();
   Session session = null;
   try {
      List<FromClause> entities = new ArrayList<FromClause>();
      entities.add(new FromClause(HQL_PROCEDIMIENTO_CLASS,
HQL_PROCEDIMIENTO_ALIAS));

      QueryBuilder qb = new QueryBuilder(
            HQL_PROCEDIMIENTO_ALIAS,
            entities,
            procedimentCriteria.getIdioma(),
            HQL_TRADUCCIONES_ALIAS,
            true);

      Restriction telematicoVigente =
ProcedimentUtils.ParseTelematicoVigente(procedimentCriteria,
HQL_PROCEDIMIENTO_ALIAS);

      if ( telematicoVigente != null )
         qb.addRestriction(telematicoVigente);

      ProcedimentUtils.parseActiu(criteris, procedimentCriteria,
HQL_PROCEDIMIENTO_ALIAS);

      criteris.addAll(BasicUtils.parseCriterias(
            ProcedimentCriteria.class,
            HQL_PROCEDIMIENTO_ALIAS,
            HQL_TRADUCCIONES_ALIAS,
            procedimentCriteria));

      qb.extendCriteriaObjects(criteris);
      session = getSession();
      Query query = qb.createQuery(session);
      numResultats = ((Integer) query.uniqueResult()).intValue();

   } catch (HibernateException e) {
      log.error(e);
      throw new EJBException(e);
   }  catch (QueryBuilderException e) {
      log.error(e);
      throw new EJBException(e);
    } catch (CriteriaObjectParseException e) {
      log.error(e);
      throw new EJBException(e);
   } finally {
      close(session);
```

```
    }
    return numResultats;
}
```

2.1.5 es.caib.rolsac.api.v2.resources:

El darrer package és el que conté els recursos compartits de l'API. Actualment només disposa de dos arxius de configuració i un de properties.

2.2 Classes generades i .class

Un cop llançat el build es generen els .class i les classes que es generen a partir dels wsdl dintre de dos nous directoris, "/output/api" i "/output/apiws/".

Dintre "/output/api/" es generen 4 directoris:

- Classes: Aquest directori conte els .class de l'API.
- gen-etc: Conté el arxius de configuració generats.
- gen-src: Conté les classes generades per els EJBs, que són les següents: XXXQueryServiceEJBHome, XXXQueryServiceRemote, XXXQueryServiceEJBUtil, XXXQueryServiceEJBSession.
- wsdl-src: El directori conté les classes del model generades a partir dels wsdl, és a dir, els XXXCriteri, XXXDTO, XXXOrdenacio, XXXQueryServiceEJBRemote, XXXQueryServiceEJBRemoteService, XXXQueryServiceEJBServiceLocator, XXXWSSoapBindingImpl i XXXWSSoapBindingStub. A més, hi ha els wsdd dels web services.

2.3 Llibreries i restriccions JBoss

Llibreries necessàries de forma directe

L'API necessita d'una serie de llibreries externes per funcionar de forma directe, a part d'aquestes, també en necessita moltes més de forma indirecte, ja que utilitza altres moduls del rolsac.

JDK 1.5:

L'API de rolsac és una aplicació compilada i preparada per funcionar amb Java 5. Això és pel fet que l'API està inclosa dintre el projecte Rolsac, el qual utilitza l'antic estàndard de la DGTIC.

EJB 2:

La utilització de la versió 2.0 als EJBs és pel fet que és l'estàndard de la DGTIC per Rolsac, igual que amb el JDK.

Axis:

Pels webs services les llibreries utilitzades han sigut les d'Axis 1.4. Això es deu al fet que és la versió més gran suportada per la versió de JBoss utilitzada.

Hibernate:

Per la part de la capa de persistència s'utilitza hibernate 2. La versió emprada és perquè és l'utilitzada al projecte Rolsac i no es plantejava migra-la.

2.4 Tasques de construcció de L'API

Dintre el projecte de Rolsac cada mòdul compta amb un build per construir els seus propis jars. L'API no n'es una excepció, i disposa d'un build.xml propi on hi ha definides les tasques per a la construcció de l'API.

Tasca clean

La tasca clean elimina o netetja els directoris explicats a l'apartat 3.2, en els quals es generen els arxius de la compilació explicats a continuació.

Tasca prepare

La primera tasca del build de l'API és la prepare que simplement genera els directoris dintre l'output.

Tasca init

La segona tasca és la que crida al xdoclet per llegir les anotacions dels EJBs.

Tasca stub.gen.all

La tercera tasca és la que genera el contingut del directori wsdl-src, és a dir, generar les classes definides als wsdl, expliades en apartats anteriors.

Tasca ejb.gen.all

La quarta tasca genera els continguts dels directoris gen-etc i gen-src que en el primer cas són els de configuració i en el segon són EJBHome, EJBRemote i EJBSession.

Tasca ejb.compile

La cinquena tasca és la que realitza la compilació del codi i per tant la que crea els .class que s'inclouen dintre el directori classes.

Tasca ejb.server, ejb.client

La sisena tasca és la de empaquetament dels .jar pròpiament dits.

Tasca main

Finalment la tasca main és la que llança les altres i és la que es cridada des del build principal del ROLSAC.

Tasques docs

A més de les tasques esmentades, existeixen 4 tasques de generació de documentació. Aquestes tasques es poden identificar per dur el sufixe "doc" al seu nom.

2.5 Configuració i KeyStore

Per la utilització de l'API mitjançant el web service és necessari utilitzar un keyStore que guarda el certificat pel tema de seguretat. Al rolsac es proporciona un keystore per provar.

Aquest KeyStore el podeu trobar a la següent ruta del projecte:

/test/api/integracion/resources/storerolsac.jks

La forma correcta d'utilitzar el KeyStore és mitjançant el mètode estàtic "autentificar" que proporciona Rolsac dintre la classe es.caib.rolsac.api.v2.general.CertificadoUtil. S'han de passar 2 paràmetres, el primer és el nom del keystore i el segon la seva clau.

I les properties del keystore proporcionat són:

- property.name=storerolsac.jks
- property.pass=contrasena

Es recomana que s'utilitzin properties en comptes de "hardcodear" el nom i la contrasenya.

A continuació es proporciona un exemple del codi d'una classe Java per utilitzar el client de l'API, però dir que és suficient amb copiar el codi, és a dir, és completament vàlid per començar a utilitzar l'API.

Amb aquesta classe no sols s'autentifica sinó que a més ens permet configurar l'estratégia de connexió a l'API (EJB/WS) mitjançant la constant API_STRATEGY.

Codi APIUtil.java

```
import es.caib.rolsac.api.v2.estadistica.EstadisticaInsertService;
import es.caib.rolsac.api.v2.general.BeanUtils;
import es.caib.rolsac.api.v2.general.BeanUtils.STRATEGY;
import es.caib.rolsac.api.v2.general.CertificadoUtil;
import es.caib.rolsac.api.v2.rolsac.RolsacQueryService;

public class APIUtil {

   public static final STRATEGY API_STRATEGY = STRATEGY.WS;

    public static enum Sexo {
        MASCULINO,
        FEMENINO;

        public static Sexo getSexo(Integer sexo) {
            return sexo == 2 ? FEMENINO : MASCULINO;
        }
    };

    private APIUtil() {}

   public static RolsacQueryService getRolsacQueryService() {

      String keyStoreName = System.getProperty("property.name");
      String keyStorePass = System.getProperty("property.pass");

      if (API_STRATEGY == STRATEGY.WS)
         CertificadoUtil.autentificar(keyStorePass, keyStoreName);

      return (RolsacQueryService) BeanUtils.getAdapter("rolsac",
APIUtil.API_STRATEGY);
```

```
    }

    public static EstadisticaInsertService getEstadisticaInsertService() {

        return (EstadisticaInsertService) BeanUtils.getAdapter("estadistica",
APIUtil.API_STRATEGY);

    }

}
```

3 Manual del desenvolupador

Tot seguit es proporciona un manual o guia ràpida per por modificar i estendre les funcionalitats de l'API.

3.1 Afegir nous mètodes WSDLs i publicar-los en el servei web

3.1.1 Lògica de negoci

Aquest pas no té a veure amb la modificació d'un WS, però per coherència el millor abans de res es crear el mètode on hi haurà la lògica, el qual es defineix al "XXXQueryServiceEJB".

3.1.2 Part client

Primer de tot s'han de modificar els WSDL on estan definits els Web Services i afegir les noves definicions als següents tags: wsdl:message, wsdl:portType, wsdl:binding i "<wsdl:types><schema>". A part d'afegir els nous nodes als diferents tags que s'exposen a continuació, també cal dir que si es necessiten nous Criterias i nous DTO també s'han d'afegir, no obstant aquest punt s'explicarà un poc més endavant.

Message

Aquí hem d'afegir els mètodes de petició i resposta que definiran el tipus de cridada. El missatge d'entrada (input) és la cridada al mètode i el valor retornat és el missatge de sortida (output).

Exemple:

- Petició: Definim el mètode i els seus arguments, per exemple si l'argument és un enter:

```
<wsdl:message name="nombreNuevoMetodoRequest">
      <wsdl:part name="in0" type="xsd:int" />
</wsdl:message>
```

- Resposta: Definim el mètode i el tipus de dada que retorna, en l'exemple es

tracte d'un objecte FitxaDTO (que ha d'estar definit a priori com un "complex type" dintre un wsdl.

```
<wsdl:message name="nombreNuevoMetodoResponse">
      <wsdl:part name="nombreNuevoMetodoReturn" type="tns3:FitxaDTO" />
</wsdl:message>
```

PortType

Dintre el següent tag (PortType) s'ha d'afegir un nou node. Dintre d'aquest tag s'han d'afegir els "operation" o nous mètodes i també els inputs (arguments de la cridada) i outputs (el retorn del servidor).

Exemple:

```
<wsdl:portType name="XXXXQueryServiceEJBRemote"><!-- Esto ya estará definido -->
...
<wsdl:operation name="nombreNuevoMetodo" parameterOrder="in0">
<wsdl:input message="impl:nombreNuevoMetodoRequest" name="nombreNuevoMetodoRequest" />
<wsdl:output message="impl:nombreNuevoMetodoResponse" name="nombreNuevoMetodoResponse" />
</wsdl:operation>
...
</wsdl:portType>
```

Binding

Finalment dintre del WSDL només ens queda afegir els bindings.

Exemple:

```
<wsdl:binding name="XXWSSoapBinding" type="impl:XXQueryServiceEJBRemote">
<wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
...
<wsdl:operation name="nombreNuevoMetodo">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="nombreNuevoMetodoRequest">
<wsdlsoap:body
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/namespace="http://intf.ejb.XX.v2.a
pi.rolsac.caib.es" use="encoded" />
</wsdl:input>
<wsdl:output name="nombreNuevoMetodoResponse">
<wsdlsoap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
namespace="http://ws.XX.v2.api.rolsac.caib.es" use="encoded" />
</wsdl:output>
</wsdl:operation>
  ...
</wsdl:binding>
```

Schema

Si s'ha utilitzat algun paràmetre Criteria o DTO i es necessita que es passin o recuperin un list/array es necessari definir un "complexType" dintre el tag "schema".

Exemple:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" …>
   <complexType name="ArrayOgFitxaDTO">
      <complexContent>
         <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="tns4:ProcedimentDTO[]"/>
         </restriction>
      </complexContent>
   </complexType>
</schema>
```

Un cop fets els canvis descrits és imprescindible realitzar un "clean" del projecte i llançar un "build" perquè es generin les classes Java pertinents. Finalment un cop acabi el "build" s'ha de revisar la classe "XXXWSSoapBindingStub" i les dues interfícies "XXXQueryServiceEJBRemote" (una al per EJB i una per el WS) i veure que el nou mètode s'ha creat.

Ruta: /output/api/wsdl-src/es/caib/rolsac/api/v2/XXX/ws/

3.1.3 Part servidor

Un cop compilada la part client es pot veure con es genera un wsdd anomenat "deploy.wsdd" on es defineixen els serveis. Es veu fàcilment que hi ha un "operation" per cada un dels mètodes del WS.

Exemple:

```
<service name="XXWS" provider="java:RPC" style="rpc" use="encoded">
    ...
<operation name="nombreNuevoMetodo" qname="operNS:doSmthg"
xmlns:operNS="http://intf.ejb.XX.v2.api.rolsac.caib.es"
returnQName="nombreNuevoMetodoReturn" returnType="rtns:long"
xmlns:rtns="http://www.w3.org/2001/XMLSchema" soapAction="" >
<parameter qname="in0" type="tns:long" xmlns:tns="http://www.w3.org/2001/XMLSchema"/>
</operation>
...
</service>
```

En aquest punt l'únic que hem de fer és copiar la nova entrada del nou mètode, es a dir, l'"operation" i aferrar-la al lloc corresponent de l'arxiu "server-config.wsdd", que és el descriptor del servei a la part servidor.

El lloc on hem d'afegir l'entrada corresponent és dintre del servei o WS que em modificat, per exemple: si toca'm el RolsacWS és aquest servei on tenim que afegir l'"operation".

Per acabar la part servidor, només hem d'afegir el nom del mètode al següent node del nostre servei:

<parameter name="allowedMethods"...

NOTA IMPORTANT: Existeixen dos arxius anomenats server-config.wsdd dins el projecte, que corresponen a les versions v1 i v2 de l'API.

API v1 --> /ws/etc/server-config.wsdd
API v2 --> /apiws/etc/server-config.wsdd

3.1.4 Modificar les classes del client

Com s'ha explicat abans, l'API utilitza diferents patrons de disseny per poder adaptar-se a diferent situacions, aquestes classes s'han de modificar per afegir el nou mètode. La forma més senzilla és copiar un mètode existent i canviar els noms, paràmetres i retorns.

Llistat de classes a modificar (XXX fa referència al WS en concret):

- XXXQueryService
- XXXQueryServiceAdapter
- XXXQueryServiceStrategy

  EJB:

  - XXXQueryServiceEJBStrategy
  - XXXQueryServiceEJBDelegate

  WS:

  - XXXQueryServiceWSStrategy
  - XXXQueryServiceWSGateway

3.2 Modificar Criterias, DTOs i Ordenacions

Els següents arxius wsdl estan a la ruta: /api/etc/wsdl/dto/*

3.2.1 Afegir nou atribut a un Criteria

Per modificar un Criteria basta anar al XXXCriteria.wsdl i afegir un node a la llista d'"element".

Exemple:

<element name="textos" nillable="true" type="xsd:string" />

3.2.2 Afegir nou atribut a un DTO

Per modificar un DTO basta anar al XXXDTO.wsdl i afegir un node a la llista d'"element".

Exemple:

<element name="familia" nillable="true" type="xsd:long" />

3.2.3 Afegir noves ordenacions per a un Criteria

Per afegir o eliminar possibles ordenacions d'un Criteria basta modificar l'arxiu XXXOrdenacio.wsdl, el qual conte un enumerat amb les possibles ordenacions. Cada un d'aquests enumerats s'ha de correspondre amb un atribut corresponent del Criteria seguit de "_" i la direcció de l'ordenació.

Exemple:

<enumeration value="fechaCaducidad_asc" />

<enumeration value="fechaCaducidad_desc" />

3.3 Afegir descriptors de nous serveis web (WSDL)

3.3.1 Crear el XXXCriteria.wsdl, XXXDTO.wsdl i el XXXOrdenar.wsdl

El primer pas és crear els wsdl dels criterias, dtos i ordenacions per generar les classes mitjançant XDoclet. El més senzill es agafar uns de ja existents i modificar els noms,

imports i posar els atributs i enumerats que ens facin falta.

3.3.2 Crear el XXXWS.wsdl

El segon pas es crear el wsdl on es descriu el WS i definir per cada nou mètode les mateixes entrades descrites al punt 4.1.2. A part d'això també s'ha d'afegir un node al final de l'arxiu, a continuació es mostra un exemple d'aquest node.

Exemple:

```
<wsdl:service name="CatalegDocumentsQueryServiceEJBRemoteService">
  <wsdl:port binding="impl:CatalegDocumentsWSSoapBinding"name="CatalegDocumentsWS">
    <wsdlsoap:address location="https://localhost:8443/sacws-
api/services/CatalegDocumentsWS"/>
  </wsdl:port>
</wsdl:service>
```

El més senzill per crear un nou wsdl és copiar-ne un d'existent i modificar-lo segons les necessitats.

3.3.3 Crear classes de la part client

El tercer pas és crear les classes Java necessàries per enllestir un nou servei de l'API.

Aquestes classes són les següents:

- XXXQueryService.java
- XXXQueryServiceAdapter.java
- XXXQueryServiceStrategy.java
- XXXQueryServiceEJBStrategy.java
- XXXQueryServiceDelegate.java
- XXXQueryServiceEJBLocator.java
- XXXQueryServiceWSStrategy.java
- XXXQueryServiceGateway.java

Como s'ha dit fins el moment, el més ràpid es copiar unes classes ja existents i modificar-les segons es requereixi.

Un comentari important es el fet que algunes d'aquestes classes requereixen que ja existeixin classes que es generen en el punt següent, per exemple: XXXQueryServiceGateway, requereix de la XXXWSSoapBindingStub. En aquests cas basta comentar les línies que criden a la classe encara no existent.

3.3.4 Build

Després de crear tots els arxius necessaris és el moment de llançar el build. Però abans s'ha de modificar el /api/buil.xml per afegir la cridada a l'AXIS a fi de generar el servei.

Exemple:

```
<axis-wsdl2java url="${ws.etc.wsdl}/EdificiWS.wsdl" output="${wsdl.gen.src}"
typemappingversion="1.1" testcase="false" verbose="true" serverside="true">
</axis-wsdl2java>
```

Un cop finalitzar veurem que s'han generat 5 classes Java, que són:

Utilitzades:

- XXXWSSoapBindingStub.java
- XXXQueryServiceEJBRemote.java

No utilitzades:

- XXXWSSoapBindingImpl.java
- XXXQueryServiceEJBService.java
- XXXQueryServiceEJBServiceLocator.java

3.3.5 Afegir el servei a la part servidor

Finalment, la darrera passa és afegir el nou servei al sever-config.wsdd. Per això hem de copiar el contingut del deploy.wsdd a l'arxiu sever-config.wsdd de manera semblant als ja existents.

4 Manual d'ús de l'API

4.1 Configuració prèvia

El primer que s'ha de fer és crear una classe que ens faciliti la configuració, el més senzill és copiar APIUtils.java proporcionat a l'apartat 3.5 (també s'explica que si es vol usar l'Api per WS és necessari un Keystore). El codi ja es proporciona 2 mètodes que creen els serveis d'estadística i el general de rolsac. Si es necessita algun més basta agafar com a referència el de Rolsac.

Exemple amb el servei de "familia":

```
public static FamiliaQueryService getFamiliaQueryService() {
    if (API_STRATEGY == STRATEGY.WS) {
        CertificadoUtil.autentificar(KeyStorePass, KeyStoreName);
    }
    return (FamiliaQueryService) BeanUtils.getAdapter("familia",
APIUtil.API_STRATEGY);
}
```

4.2 Criteria i ordenació

Un cop tenim la configuració bàsica que ens facilitarà les coses podem començar a definir el filtratge de les consultes.

Per això es defineix el Criteria necessari pel mètode a utilitzar i es configuren els criteris de la cerca. Amb això es vol dir que si utilitza'm un mètode d'un servei que per exemple ens llista seccions i aquest mètode requereix un SeccioCriteria podrem filtrar o restringir la cerca per les diferents propietats del SeccioCriteria.

Exemple:

```
SeccioCriteria seccionCriteria = new SeccioCriteria();
seccionCriteria.setIdioma("ca");
seccionCriteria.setCodigoEstandard("seu");
```

A part, també es pot recuperar tota aquesta informació de forma ordenada mitjançant el mètode ".setOrdenar(...)" dels diferents Criterias.

A n'aquest mètode s'ha de passar un array d'ordenacions predefinides per cada

Criteria. Existeix una classe d'ordenacions per cada tipus de Criteria que permeti ordenacions. La prioritat de l'ordenació és la mateixa que l'ordre de l'array.

Exemple:

```
IniciacioOrdenacio[] ordenacions = new IniciacioOrdenacio[2];
ordenacions[0] = IniciacioOrdenacio.codigoEstandar_asc;
ordenacions[1] = IniciacioOrdenacio.id_desc;
iniciacioCriteria.setOrdenar(ordenacions);
```

4.3 Consulta

Per realitzar la consulta sols és necessari cridar al mètode desitjat amb el Criteria adient. Tot seguit hi ha un exemple d'un mètode fent ús de l'API.

Exemple complet:

```
public List<SeccioQueryServiceAdapter> getSecciones(List<String>
codigosEstandar, String idioma) throws ServiceException {

    RolsacQueryService rqs = APIUtil.getRolsacQueryService();
    List<SeccioQueryServiceAdapter> seccionList = new
ArrayList<SeccioQueryServiceAdapter>();

    for (String cod: codigosEstandar) {
       SeccioCriteria seccionCriteria = new SeccioCriteria();
       seccionCriteria.setIdioma("ca");
       seccionCriteria.setCodigoEstandard("seu");
        seccionCriteria.setOrdenar(new SeccioOrdenacio[] {
SeccioOrdenacio.perfil_desc });

       try {
          seccionList.add(rqs.llistarSeccions(seccionCriteria).get(0));
        } catch (QueryServiceException e) {
            throw new ServiceException(e);
        }
    }
    return seccionList;
}
```

4.4 Serveis disponibles

Actualment l'API disposa de 34 serveis com ja s'ha exposat. Tots els serveis s'utilitzen de la mateixa forma com s'ha dit en aquest apartat 5. A continuació es llisten els serveis disponibles:

- AgrupacioFetVitalWS
- AgrupacioMateriaWS

- ButlletiWS
- CatalegDocumentsWS
- DocumentTramitWS
- DocumentWS
- EdificiWS
- EnllacWS
- EspaiTerritorialWS
- EstadisticaWS
- ExcepcioDocumentalWS
- FamiliaWS
- FetVitalWS
- FitxaUAWS
- FitxaWS
- FormulariWS
- IconaFamiliaWS
- IconaMateriaWS
- IdiomaWS
- MateriaAgrupacioWS
- MateriaWS
- NormativaWS
- PerfilWS
- PersonalWS
- ProcedimentWS
- PublicObjectiuWS

- RolsacWS
- SeccioWS
- TaxaWS
- TipusWS
- TramitWS
- UnitatAdministrativaWS
- UnitatMateriaWS
- UsuariWS