

# Estándar de aplicaciones Java EE

## Estándares de desarrollo

Palma, septiembre de 2025



**Vicepresidència Primera i Conselleria  
d'Economia, Hisenda i Innovació**  
Direcció General d'Estratègia Digital  
i Desenvolupament Tecnològic



## Índice

<b>HISTORIAL DE VERSIONES.....</b>	<b>3</b>
<b>1. INTRODUCCIÓN.....</b>	<b>4</b>
<b>2. NORMAS DE CARÁCTER GENERAL.....</b>	<b>5</b>
2.1. Especificaciones tecnológicas.....	7
<b>3. ESTRUCTURA DEL PROYECTO.....</b>	<b>10</b>
<b>4. NORMATIVA.....</b>	<b>16</b>
4.1. Normativa general de los ficheros.....	16
4.2. Nomenclatura.....	17
4.2.1. <i>Nomenclatura y jerarquía de paquetes.....</i>	<i>17</i>
4.2.2. <i>Nomenclatura de clases.....</i>	<i>18</i>
4.2.3. <i>Nomenclatura de métodos.....</i>	<i>18</i>
4.2.4. <i>Nomenclatura de atributos y variables.....</i>	<i>19</i>
4.2.5. <i>Nomenclatura de constantes.....</i>	<i>19</i>
4.2.6. <i>Nomenclatura de servicios e interfaces.....</i>	<i>19</i>
4.2.7. <i>Nomenclatura de tests unitarios.....</i>	<i>20</i>
4.2.8. <i>Servicios de directorio de servidor de aplicaciones.....</i>	<i>20</i>
4.2.9. <i>Acceso a bases de datos.....</i>	<i>20</i>
4.3. Versionado de código.....	22
4.3.1. <i>Mostrar la versión del producto durante la ejecución del producto.....</i>	<i>22</i>
4.3.2. <i>Crear plantilla de la clase del versionado.....</i>	<i>22</i>
4.3.3. <i>Hacer referencia a la versión en las páginas o clases Java.....</i>	<i>23</i>
4.3.4. <i>Mostrar la versión en el log de aplicación.....</i>	<i>23</i>
4.4. Propiedades de configuración.....	23
4.5. Implantación de buenas prácticas en el código.....	25
4.6. Restricciones adicionales.....	25
<b>5. SEGURIDAD.....</b>	<b>27</b>
5.1. Autenticación.....	27
5.2. Control de acceso al módulo web.....	29
5.3. Protección de EJBs.....	31
5.4. Dominios de seguridad.....	32
<b>6. SERVICIOS WEB Y API REST.....</b>	<b>33</b>



## Historial de versiones

Fecha	Versión	Descripción	Autor
05/04/16	7.2	Revisión de estándares	DGDT
17/01/20	9.0	Cambio formato documento Revisión de estándares	DGMAD
07/02/20	9.1	Corrección de errores y aclaraciones	DGMAD
22/07/20	9.2	Eliminar versionado de los war y jar	DGMAD
03/12/20	9.5	Aclaraciones	DGMAD
23/03/21	9.6	Validación del framework Spring Separación de contextos API REST interna y externa	DGMAD
24/01/22	9.7	Corrección de errores y aclaraciones	DGEDDT
08/09/22	9.8	Corrección error datasource	DGMAD
19/09/22	9.9	Otras correcciones y aclaraciones	DGMAD
25/05/23	9.10	Nueva sección de reutilización de datos	DGMAD
07/07/23	9.11	Corrección de errores en el ejemplo del datasource	DGMAD
19/09/25	9.12	Revisión de formato y separación de la sección «7. Servicios Rest de reutilización de datos» en un nuevo documento	DGEDDT



## 1. INTRODUCCIÓN

Este documento detalla el estándar de desarrollo de aplicaciones Java Platform Enterprise Edition (Java EE) que se debe seguir para el desarrollo de aplicaciones que se instalarán en los servidores de la Dirección General de Estrategia Digital y Desarrollo Tecnológico (DGEDDT).

El documento se estructura en cinco grandes apartados:

1. Normas de carácter general (Capítulo 2).
2. Estructura del proyecto (Capítulo 3)
3. Normativa y recomendaciones sobre el código (Capítulo 4).
4. Aspectos de seguridad (Capítulo 5).
5. Servicios API REST (Capítulo 6)

Cualquier tecnología, *framework* o decisión técnica que salga de estos estándares de desarrollo deberán ser consultados y aprobados por la DGEDDT. En cualquier caso, si no se realiza dicha consulta, la DGEDDT se reserva el derecho de no aceptar el desarrollo o exigir el cumplimiento de estos estándares.

### **Nota:**

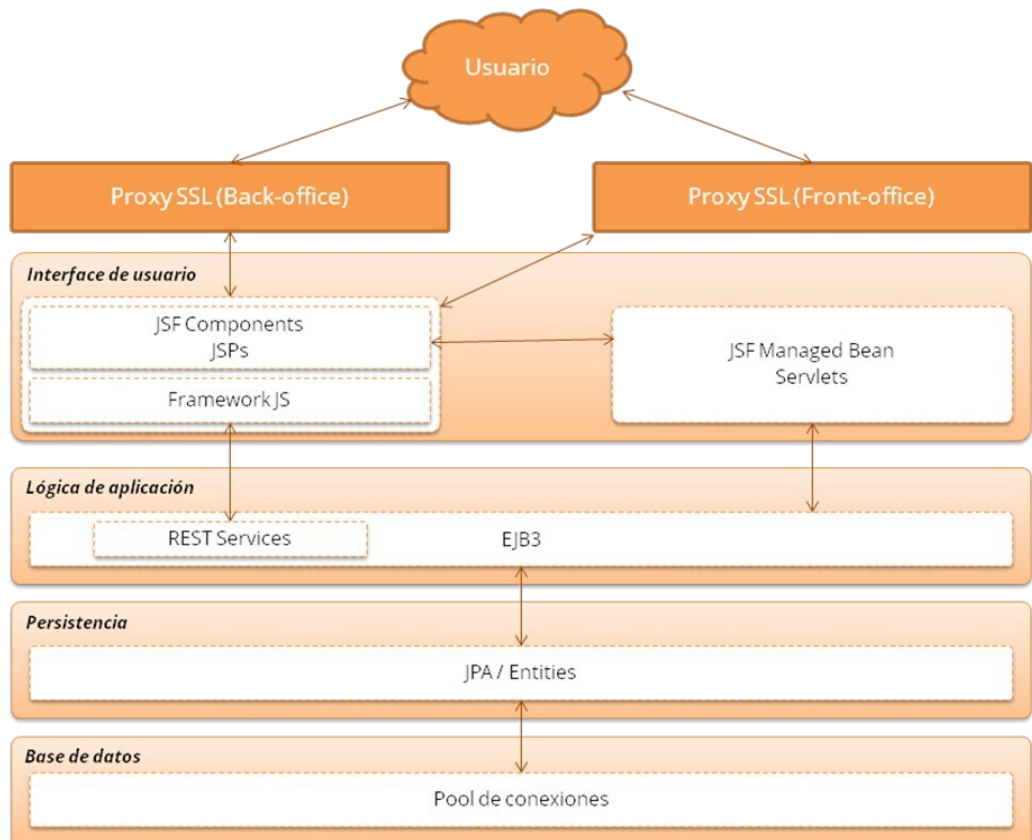
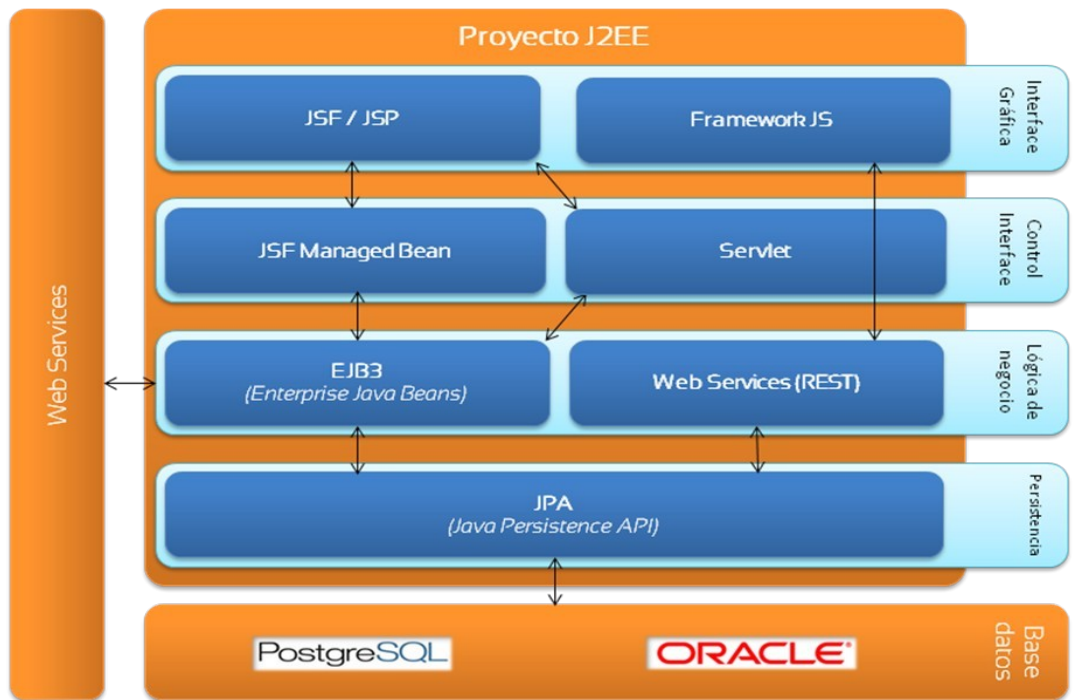
**Este documento es una traducción del documento original en catalán. En caso de conflicto prevalecerá la versión catalana del mismo.**



## 2. Normas de carácter general

A continuación se detallan un conjunto de normas de carácter general relacionadas con el desarrollo de aplicaciones Java EE.

- Además de las especificaciones descritas en este documento, las aplicaciones se desarrollarán siguiendo las especificaciones del resto de documentos de los estándares de desarrollo del GOIB, especialmente las especificaciones descritas en:
  - El estándar de implantación de aplicaciones.
  - El estándar de base de datos.
  - El estándar de reutilización y apertura de datos.
- Las aplicaciones deberán desarrollar los módulos mediante **aplicaciones distribuidas** en tres niveles: interfaz, lógica, y datos; donde la capa de presentación se puede subdividir en capa de interfaz gráfica y capa de control de interfaz, y la capa de datos en capa de persistencia y capa de base de datos.
- Las aplicaciones utilizarán el **patrón de diseño Modelo-Vista-Controlador (MVC)**. La petición del usuario será recibida por el controlador de la capa de control de interfaz, el cual accederá a la capa lógica de negocio implementado por los Enterprise Java Bean (EJBs) y los servicios REST. Los EJBs accederán a las entidades de persistencia gestionadas por JPA.
- **Bajo ninguna circunstancia ni los controladores, ni los servlets, ni las páginas JSP/JSF/HTML accederán de forma directa a la base de datos.**





- Todo proyecto deberá publicarse en un repositorio git del GOIB según su tipología:
  - Aplicaciones de código libre: <https://github.com/GovernIB/>
  - Aplicaciones internas: <https://git.caib.es/>
- El producto final y las actualizaciones se entregarán según el formulario estándar de cuadernos de carga estandarizados por la DGEDDT (ver Documento “Estándar de implantación de aplicaciones”, Capítulo 4. “Cuaderno de carga”).
- El sistema deberá cumplir las medidas y criterios de seguridad designadas en la legislación vigente, especialmente en:
  - La **Ley Orgánica 3/2018**, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales, junto al Reglamento (UE) 2016/679 del parlamento europeo y del consejo de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento General de Protección de Datos).
  - El **Real Decreto 3/2010**, de 8 de enero, por el que se regula el **Esquema Nacional de Seguridad** en el ámbito de la Administración Electrónica (ENS).
  - La **Ley 34/2002**, de 11 de julio, de servicios de la sociedad de la información y de comercio electrónico (LSSI).
- Así mismo, el sistema deberá cumplir las medidas de accesibilidad del **Real Decreto 1112/2018**, de 7 de septiembre, sobre **accesibilidad** de los sitios web y aplicaciones para dispositivos móviles del sector público, junto con las WCAG (Web Content Accessibility Guidelines) en su versión 2.1.

## **2.1. Especificaciones tecnológicas.**

A **nivel general**, las aplicaciones tendrán que implementarse utilizando la versión libre de la plataforma de desarrollo de Java **OpenJDK 11** y



ejecutarse sobre un servidor de aplicaciones **JBoss EAP 7.2** sin modificar (exceptuando los parámetros descritos en este documento).

Respecto a la **capa de presentación**:

- Se utilizará preferentemente **JSF 2.3** (Java Server Pages) junto con la biblioteca de componentes **Primefaces**.
- El punto anterior podrá sustituirse por un framework que utilice el patrón de diseño Modelo-Vista-Controlador (MVC) y que haya sido validado previamente por la DGEDDT; actualmente: **Spring**, **Angular**, o **React**.
- En ningún caso se podrá utilizar un framework propietario o que requiera licencia.
- La interfaz de usuario tendrá que ser compatible con la última versión estable de los navegadores **Firefox ESR** y **Chrome**.

Respecto a la **capa de lógica de negocio**:

- Se utilizará preferentemente **EJBs 3.2** y servicios REST.
- Los EJBs podrán sustituirse por los módulos correspondientes de los frameworks utilizados en la capa de presentación.

Respecto a la **capa de persistencia de datos**:

- Las aplicaciones tendrán que estar preparadas para funcionar sobre una base de datos **Oracle 19.3**.
- La necesidad de utilización otras bases de datos como **PostgreSQL 10** tendrá que venir motivada y ser aprobada por parte del responsable del área de Administración de Base de datos a causa de las limitaciones de apoyo y alta disponibilidad.
- El acceso a los datos se llevará a cabo a través de la API **JPA 2.2**.
- El punto anterior podrá sustituirse por el módulo correspondiente del framework utilizado en la capa de presentación (como por ejemplo Spring Data JPA incluido en Spring).



Respecto a la **autenticación de usuarios**:

- Si son usuarios internos del GOIB, se realizará mediante el mecanismo de autenticación centralizado RedHat Single Sign-Donde 7.3 basado en **Keycloak** (ver sección «5. Seguridad»).
- Cualquier otra autenticación externa se realizará mediante el componente horizontal **LoginIB** (ver documento «Catálogo de Servicios»).

Por último, todo proyecto Java EE deberá estar desarrollado utilizando **Maven 3.6 o superior** para la construcción del proyecto (compilación y empaquetado) y la gestión de dependencias.

En función de criterios de mantenimiento y disponibilidad de versiones, y con el objetivo de mejorar el servicio ofrecido a las consejerías, el Centro de Proceso de Datos de la DGEDDT se reserva la facultad de actualizar las versiones del software aquí expuestas por otras superiores en el momento de la puesta en producción.

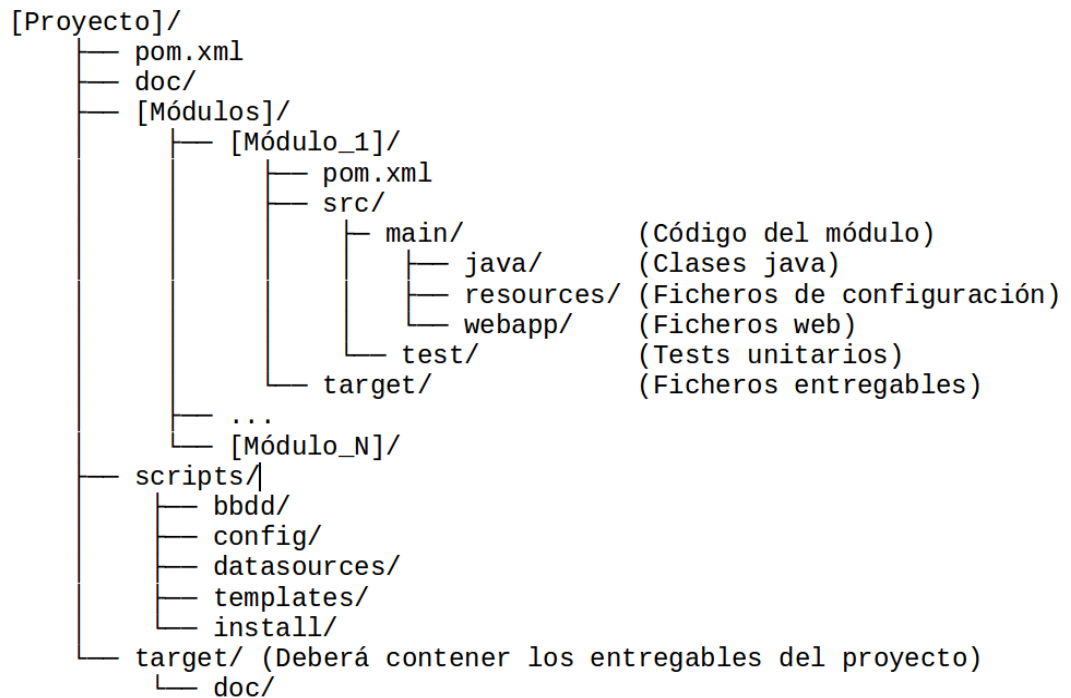
La siguiente imagen resume las especificaciones descritas en función del nivel de ejecución.

<i>Interfície d'usuari</i>	
<i>Producte</i> Firefox ESR Chrome	<i>Tecnologia</i> JSF 2.3 amb Primefaces Spring, Angular, React Altres frameworks JS (* consultar)
<i>Lògica de negoci</i>	
<i>Producte</i> JBoss EAP 7.2 OpenJDK 11	<i>Tecnologia</i> EJB 3.2 Spring, Angular, React Serveis REST
<i>Persistència de dades</i>	
<i>Producte</i> Oracle 19.3 (o superior) PostgreSQL 10 (* consultar)	<i>Tecnologia</i> JPA 2.2 JDBC 8



### 3. Estructura del proyecto

Partiendo de la base que todos los proyectos GOIB deberán utilizar Maven, la estructura a la hora de definir los proyectos será la siguiente:



#### [Proyecto]

El módulo padre del resto de submódulos Maven del proyecto. En nombre de la carpeta deberá ser el nombre del proyecto. Por ejemplo: projectbase.

#### [Proyecto]/pom.xml

Fichero Maven encargado de la configuración, compilación y empaquetado de todo el proyecto. Se encargará de ejecutar los ficheros pom.xml de los submódulos del proyecto y dejar los ficheros entregables en la carpeta [Proyecto]/target.

El fichero pom.xml del módulo padre deberá tener packaging tipo POM y deberá de contener al resto de módulos. Ejemplo de fichero POM del projectbase:



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>es.caib.projectebase</groupId>
<artifactId>projectebase</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>pom</packaging>
<name>Projecte Base</name>
<description>Projecte Base con OpenJDK 11</description>
[...]
<properties>[...]</properties>
[...]
<modules>
<module>projectebase-api</module>
<module>projectebase-back</module>
<module>projectebase-commons</module>
<module>projectebase-ear</module>
<module>projectebase-ejb</module>
<module>projectebase-front</module>
<module>projectebase-persistence</module>
<module>projectebase-service</module>
</modules>
[...]
</project>
```

### [Proyecto]/README.md

Documento sencillo donde se expliquen los requisitos, configuraciones y acciones para poder compilar, probar e instalar el proyecto.

### [Proyecto]/doc

En esta carpeta se ubicarán los ficheros de documentación del proyecto. Documentos tales como diagramas, documentos de administración (instalación, configuraciones o relaciones con otros productos...), manuales de usuario, etc. Preferiblemente estarán en formato *LibreOffice*. También se deberán ubicar en esta carpeta los fuentes utilizados para la generación de los documentos, imágenes, gráficos, formatos nativos...



La documentación específica de cada submódulo, en caso de ser necesaria, se ubicará en subcarpetas con el mismo nombre que los submódulos.

### [Proyecto]/doc/pdf

Aquí se dejarán los documentos anteriores convertidos en PDF. Son las versiones finales de los documentos.

### [Proyecto]/[Módulos]/[Módulo N]

Los proyectos tendrán que estar formados (como mínimo) por los siguientes módulos:

- a) Un **módulo padre** con packaging tipo POM que tendrá que contener al resto de módulos.
- b) Uno o varios **módulos Web (Frontal / Backoffice)** con packaging WAR que contengan todo el código de la capa de presentación: control de interfaz e interfaz. Los nombres de los ficheros generados serán códigoAplicación-front.war y códigoAplicación-back.war.
- c) Un módulo con la **lógica de negocio** y un módulo con la capa de persistencia con packaging JAR. El nombre del fichero generado será códigoAplicación.jar.
- d) Un módulo de empaquetado del resto de módulos con packaging tipo **EAR**. El nombre del fichero generado será códigoAplicación.ear y contendrá el resto de ficheros de la aplicación, incluidos el ficheros war y jar anteriores.

La nomenclatura de los módulos MAVEN será la siguiente:

- *códigoAplicación-**api**: servicios REST.*
- *códigoAplicación-**back**: interfaz web para el back-office.*
- *códigoAplicación-**front**: interfaz web para el front-office.*
- *códigoAplicación-**commons** clases y métodos con funcionalidad común a todos los módulos.*



- *códigoAplicación-ear*: capa para el empaquetado del resto de módulos (EAR).
- *códigoAplicación-service*: capa lógica de negocio.
- *códigoAplicación-ejb*: capa de Enterprise Java Bean.
- *códigoAplicación-persistence*: capa para almacenar los datos de forma persistente.
- *códigoAplicación-lib*: repositorio local de librerías externas. Utilizando *Maven* no deberíamos utilizar este módulo; si bien es cierto que algunas librerías no están disponibles en repositorios externos y en ocasiones no queda más remedio que añadirlas manualmente.

#### **[Proyecto]/[Módulos]/[Módulo N]/pom.xml**

Fichero *Maven* encargado de configurar, compilar y empaquetar el módulo.

#### **[Proyecto]/[Módulos]/[Módulo N]/src**

Contendrá el código Java del módulo (*/main/java*), el código web (*/main/webapp*), el código para testear el código Java (*/test*) y todos recursos necesarios para configurar el módulo (*/main/resources*).

#### **[Proyecto]/[Módulos]/[Módulo N]/src/main/java**

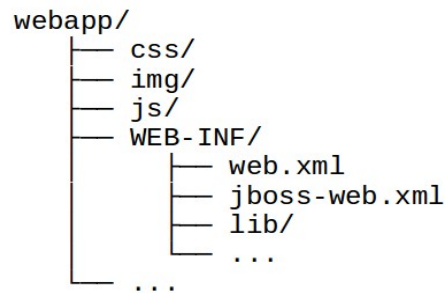
Directorio donde se ubicará el código fuente del módulo. La estructura de la paquetería viene definida en el punto 2.1. *Packages*.

#### **[Proyecto]/[Módulos]/[Módulo N]/src/main/resources**

Contendrá los recursos necesarios para la aplicación. Archivos tales como: *properties*, *xml*, imágenes,...

#### **[Proyecto]/[Módulos]/[Módulo N]/src/main/webapp**

Esta carpeta contendrá todos los ficheros necesarios para especificar la *interface* web del proyecto. Estará formado principalmente por ficheros *JSF*, *JSP*, plantillas *HTML*, *CSS*, *Javascript*, imágenes, ficheros de configuración, etc. La estructura básica del módulo web deberá ser:



### **[Proyecto]/[Módulos]/[Módulo N]/src/test/**

Este directorio contendrá los tests unitarios a realizar para probar el código ubicado en “[Proyecto]/[Módulos]/[Módulo N]/src/main/java”.

### **[Proyecto]/[Módulos]/[Módulo N]/target**

En este directorio se guardarán los ficheros resultados de la compilación y empaquetamiento del módulo.

### **[Proyecto]/scripts/**

Contendrá los *scripts* de BBDD de la aplicación. Normalmente de configuración e instalación, *datasources*, procesamiento de datos, etc.

#### **[Proyecto]/scripts/bbdd/**

Este directorio contendrá todos los *scripts* de BBDD, tanto para la creación y actualización de los objetos (sentencias DDL), como los *scripts* de inserción / eliminación / actualización de datos (sentencias DML). El contenido y estructura de estos *scripts* seguirán el formato del Capítulo 3.5 del documento “Estándar de implantación de aplicaciones”. El contenido de los *scripts* deberá cumplir las normas establecidas en el documento “Estándar de base de datos”.

#### **[Proyecto]/scripts/bbdd/completo**

*Scripts* de creación de la BD desde cero a la última versión.

#### **[Proyecto]/scripts/bbdd/[versión]**

*Scripts* incrementales para la actualización de la BD desde la versión inmediatamente anterior a la versión [versión].

#### **[Proyecto]/scripts/config/**

*Scripts* de configuración del sistema y del producto.



### **[Proyecto]/scripts/datasources/**

En esta carpeta se ubicarán los ficheros utilizados por *JBoss* para acceder a la BBDD. Dichos ficheros contienen la información necesaria para conectarse a una BBDD: *driver* de conexión, tipo de BD, *host*, puerto, nombre de la BBDD, usuario, contraseña, etc.

### **[Proyecto]/scripts/templates/**

En este directorio se guardarán las plantillas para generar ficheros. Por ejemplo, para generar los ficheros de versión que se explican en el punto 4.3.

### **[Proyecto]/target/**

Directorio donde se ubicará el producto final encapsulado y listo para el despliegue. En esta carpeta deberá haber el/los ficheros *EAR* o *JAR* a desplegar.

### **[Proyecto]/target/doc**

En esta carpeta se deberán ubicar los documentos asociados al proyecto empaquetado. Por ejemplo:

- *javadocs*: Documentos generados a partir de las clases Java. Es por ello que será imprescindible documentar correctamente cada clase siguiendo dicha especificación.
- Otros documentos que se considere necesario.



## 4. Normativa

### 4.1. Normativa general de los ficheros

Se deberá seguir:

- La codificación de los ficheros deberá ser siempre **UTF-8**.
- Se tiene que documentar cada fichero Java según la especificación JavaDoc. Antes de la declaración de las clases, se informará de la utilidad de las clases, autores y versiones. En los métodos se indicará su utilidad, las variables de entrada y salida, y las posibles excepciones.

Por otro lado, se recomienda:

- **Indentación:** No usar tabulaciones. La tabulación se realizaría mediante cuatro espacios. Se pueden configurar los IDEs de programación para que emulen la tabulación.
- **Caracteres por línea:** No superar los 120 caracteres por línea.
- **Líneas por fichero:** No exceder las 1000 líneas de código por fichero. El objetivo de esta medida es distribuir el código de forma equitativa en los ficheros y dividir conceptualmente el código de forma más efectiva.
- **XML:** Situado al inicio del documento, con una descripción de la utilidad del fichero y en cada bloque de datos la finalidad del mismo.
- **Corchetes:** Abrir corchete en la misma línea que el comando y cerrar en una línea posterior. Ejemplo de utilización:

```
public class Exemple {  
    public void getMetode(){  
        try {  
            for(int x=0; x < 10; x++){  
                while(...) {  
                    if (x < 5) {  
                        ...  
                    } else {  
                        ...  
                    }  
                }  
            }  
        }  
    }  
}
```



```
    } // Final if-else
  } // Final while
} // Final for
} catch(Exception e) {
  do {
    switch(...) {
      ...
    } // Final switch
  } while(...); // Final do-while
} // Final try-catch
} // Final método
} // Final classe
```

## 4.2. Nomenclatura

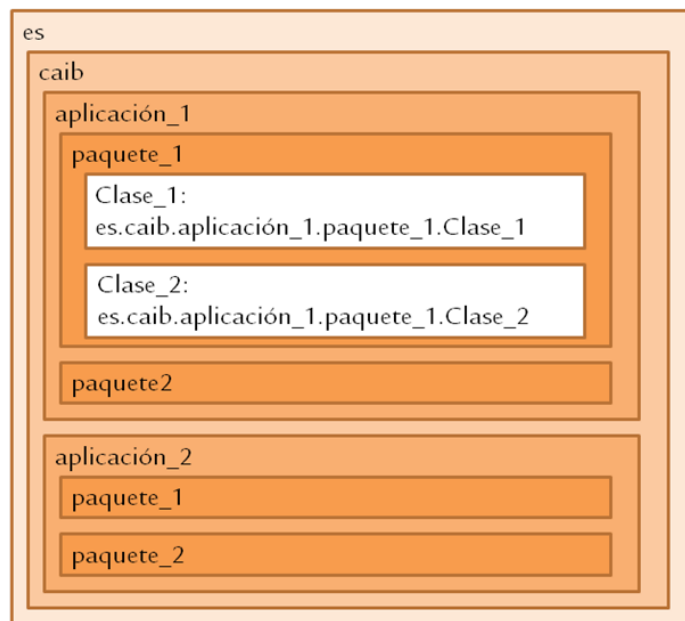
### 4.2.1. Nomenclatura y jerarquía de paquetes

Las clases de objetos se estructurarán en paquetes (packages).

Los nombres de los paquetes estarán siempre en minúsculas y podrán ser nombrados, dentro del paquete de aplicación, a criterio de analistas y diseñadores.

Todas las aplicaciones tendrán que tener como mínimo cuatro niveles: los dos primeros niveles serán siempre: **es.caib**; el tercer nivel indicará el nombre de la aplicación; y el cuarto nivel el módulo. Así, las clases se denominarán `es.caib.códigoAplicación.paquete.Clase`, tal como puede observarse en la siguiente ilustración:

El código de aplicación será suministrado por el Centro de Procesamiento de Datos de la DGEDDT (ver documento «Estándar de implantación de aplicaciones», Capítulo 2. «Solicitud de código de aplicación»).



#### 4.2.2. Nomenclatura de clases

Respecto al nombre de las clases:

- Estará formado por una o varias palabras concatenadas que describan la funcionalidad de la clase.
- La primera letra de cada palabra tiene que estar en mayúscula y el resto en minúsculas.
- No tienen que aparecer guiones.
- Tiene que tener más de un carácter.

Ejemplos: Empresa, Cliente, CentrosDeTrabajo, PersonaPorMunicipio,...

#### 4.2.3. Nomenclatura de métodos

Respecto al nombre de los métodos:

- Estará formado por una o varias palabras concatenadas que describan la funcionalidad del método.
- La primera palabra tiene que estar toda en minúsculas.
- La primera letra de cada palabra interna (a partir de la segunda) tiene que estar en mayúscula.
- No tienen que aparecer guiones.



- Se tienen que seguir la convención de Java de un prefijo más un nombre descriptivo. El prefijo (opcional si el nombre es muy simple) será: «is», «get», «set», «add», «del», «init», «close»,..
- Tiene que tener más de un carácter.

Ejemplos: connect(), isAdministrator(), getState(), addUser(), initDatos(), connect(),..

#### **4.2.4. Nomenclatura de atributos y variables**

Respecto al nombre de los atributos y variables:

- Estará formado por una o varias palabras concatenadas que describan la funcionalidad de la variable.
- La primera palabra tiene que estar toda en minúsculas.
- La primera letra de cada palabra interna (a partir de la segunda) tiene que estar en mayúscula.
- No tienen que aparecer guiones.
- Tiene que tener más de un carácter.

Ejemplos: total, municipio, nombreSocial, fechaNotificacion,...

#### **4.2.5. Nomenclatura de constantes**

Respecto al nombre de las constantes:

- Estará formado por una o varias palabras concatenadas que describan la funcionalidad de la constante.
- Todas las palabras en mayúsculas.
- Guiones bajos para separar palabras.
- Tiene que tener más de un carácter.

Ejemplos: ANCHURA, URL\_SERVIDOR, USUARIO\_BD,...

#### **4.2.6. Nomenclatura de servicios e interfaces**

Respecto al nombre de los servicios e interfaces (interface):



- Se seguirá la misma nomenclatura que el resto de clases, añadiendo al final el literal «EJB» para la implementación del servicio y el literal «Service» para la definición de la interfaz.

Ejemplos: EmpresaEJB y EmpresaService, CentrosDeTrabajoEJB y CentrosDeTrabajoService.

#### **4.2.7. Nomenclatura de tests unitarios**

Respecto al nombre de las macetas unitarias:

- Se seguirá la misma nomenclatura que el resto de clases, añadiendo al final el literal «Test».

Ejemplos: EmpresaService y EmpresaServiceTest, CentrosDeTrabajoService y CentrosDeTrabajoServiceTest.

#### **4.2.8. Servicios de directorio de servidor de aplicaciones**

El acceso al servicio de directorio (*NamingFactory*) se realizará siempre con los parámetros por defecto, asumiendo que las propiedades *JNDI* están correctamente configuradas.

Los servicios de directorio del servidor transaccional identificarán cada *Enterprise Java Bean* mediante su nombre jerárquico completo, debiendo acceder las clases Java a él mediante dicho nombre.

#### **4.2.9. Acceso a bases de datos**

El acceso a bases de datos se realizará a través de la capa de persistencia, definida en el fichero ***persistence.xml***. Dicho acceso se realizará mediante la unidad de persistencia definida en dicho fichero.

A continuación se muestra un ejemplo completo de fichero «*persistence.xml*» para la aplicación «*projectbaseexemple*».



```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd" version="2.2">
  <persistence-unit name="projectbaseexemplePU" transaction-type="JTA">
    <jta-data-source>java:jboss/datasources/projectbaseexempleDS</jta-data-source>

    <!-- Lista de clases persistentes -->
    <class>es.caib.projectbaseexemple.persistence.UnitatOrganica</class>
    <class>es.caib.projectbaseexemple.persistence.Procediment</class>

    <!-- Lista de propiedades -->
    <properties>
      <property name="hibernate.dialect" value="org.hibernate.dialect.OracleDialect"/>
      <property name="hibernate.showSql" value="false1"/>
    </properties>
  </persistence-unit>
</persistence>
```

Los datos de conexión (servidor, puerto, usuario, contraseña,..) se definirán en un fichero **datasource** independiente con nombre **códigoAplicación-ds.xml** (por ejemplo: projectbaseexemple-ds.xml).

Dentro del datasource, se tiene que definir los siguientes parámetros:

- El código del datasource para registrarlo en la interfaz de nombres de directorio de Java (**jndi-name**). Este código ha de seguir la nomenclatura «java:jboss/datasources/códigoAplicaciónDS» (por ejemplo, ««java:jboss/datasources/projectbaseexempleDS»»)
- La dirección de conexión (**connection-url**) con: el tipo de cliente, el servidor, el puerto, y el nombre de la instancia de la base de datos -SID- (por ejemplo: «jdbc:oracle:thin:@localhost:1521:projectbaseexemple»).
- El tipo de controlador (**driver**): oracle o postgresql.
- Las credenciales (**security**) del usuario (user-name y password) de base de datos. El nombre tendrá que seguir la nomenclatura «WWW\_CODIGOAPLICACION».

<sup>1</sup> En entornos de producción esta propiedad ha de estar definida siempre a «false».



- El esquema por defecto (**new-connection-sql**).

A continuación se muestra un ejemplo completo de fichero `projectbaseexemple-ds.xml` para la aplicación «`projectbaseexemple`».

```
<datasource jndi-name="java:jboss/datasources/projectbaseexempleDS" pool-
name="projectbaseexempleDS">
<connection-url>jdbc:oracle:thin:@localhost:1521:projectbaseexemple</connection-url>
  <driver>oracle</driver>
  <security>
    <user-name>WWW_PROJECTBASEEXEMPLE</user-name>
    <password>contraseña</password>
  </security>

  <new-connection-sql>
  BEGIN
  EXECUTE IMMEDIATE 'ALTER SESSION SET CURRENT_SCHEMA =
PROJECTBASEEXEMPLE';
  END;
  </new-connection-sql>

</datasource>
```

### 4.3. *Versionado de código*

#### 4.3.1. *Mostrar la versión del producto durante la ejecución del producto*

Para mostrar la información de versión de la aplicación, en *logs*, páginas, etc. Será obligatorio generar clases necesarias para ello:

#### 4.3.2. *Crear plantilla de la clase del versionado*

Crear el fichero ***Version.java.template*** en `[Proyecto]/scripts/templates` con el siguiente contenido:

```
package es.caib.projectbase;
// Código autogenerado por la Version.java.template.
public final class Version {
  public static final String VERSION="@project.version@";
  public static final String BUILDTIME="@project.buildtime@";
  public static final String VCS_REVISION="@vcs.revision@";
  public static final String JDK_VERSION="@jdk.version@";
}
```



### 4.3.3. Hacer referencia a la versión en las páginas o clases Java

Para hacer referencia a la versión y fecha de generación en las páginas se puede hacer invocando algún *Servlet*, Controlador *JSF*, utilizando `<%=Version.VERSION%>` en *JSP*, etc, que utilice la clase *Version.java* generada anteriormente y que introduzca en el pie de la página la versión y la fecha de generación del producto.

En cuanto a las referencias en clases java se podrá hacer utilizando *Version.VERSION*, siempre y cuando se haya definido la clase *Version.java* a partir de la plantilla *Version.java.template*.

### 4.3.4. Mostrar la versión en el log de aplicación

Al arrancar la aplicación de deberá imprimir un log (utilizando el *logger* a nivel *INFO*) con el nombre del producto y la versión que se está ejecutando así como la fecha de generación. Por ejemplo:

```
LOGGER.info("Cargando la aplicación PROYECTOBASE versión "+Version.VERSION+ "  
generada en fecha: "+Version.BUILDTIME);
```

## 4.4. Propiedades de configuración

Las aplicaciones podrán hacer uso de propiedades para parametrizar diferentes aspectos de la aplicación mediante el uso de dos ficheros *properties*:

- ***códigoAplicación.properties***: de propiedades internas de cada aplicación. La DGEDDT en ningún caso se hará cargo de añadir propiedades o modificar los valores de este fichero. Se deberá mandar a despliegue el fichero entero que ya incorpore los cambios respecto al anterior.
- ***códigoAplicación.system.properties***: de propiedades configurables por parte de la DGEDDT. En este caso, la DGEDDT sí que se encargará de configurar dicho fichero, que se usará exclusivamente para las siguientes propiedades (de uso opcional):
  - ***es.caib.códigoAplicación.fitxers***: previa solicitud, apuntará a un directorio en el cual la aplicación podrá guardar ficheros. En dicho directorio, la aplicación podrá crear la estructura que mejor



se adapte a sus necesidades. Si fuera necesario crear una estructura con diferentes subdirectorios, éstos deberán crearse bajo esa ruta, y se deberán definir las propiedades necesarias relativas al directorio base apuntado por `es.caib.códigoAplicación.fitxers` en el fichero `códigoAplicación.properties`.

- **es.caib.códigoAplicación.int.nombreIntegración.usuario** y
- **es.caib.códigoAplicación.int.nombreIntegración.secret**: para los casos en que la aplicación tenga que realizar llamadas a webservices, la DGEDDT configurará el usuario y el password, respectivamente en dichas propiedades. En el caso de usuarios de aplicación, `nombreIntegración` tiene el formato `aplicacionOrigen_aplicacionDestino`.
- **es.caib.códigoAplicación.int.nombreIntegración.endpoint**: Url que apunta al servicio de aplicación que se desea llamar.
- **es.caib.códigoAplicación.int.nombreIntegración.path**: Path hacia la integración deseada. Ej: path hacia ruta de ficheros estáticos: `es.caib.codigo.int.staticfiles.path=C:/Desarrollo`.

Las aplicaciones podrán cargar los ficheros de *properties* a través de las propiedades de sistema:

- `es.caib.códigoAplicación.properties`
- `es.caib.códigoAplicación.system.properties`

A continuación se muestra un ejemplo de código para cargar las propiedades de los diferentes ficheros *properties*:

```
try (InputStream input = new
FileInputStream(System.getProperty("es.caib.codigoAplicacion.properties")) {
    Properties prop = new Properties();

    // carga propiedades
    prop.load(input);

    // obtiene el valor de cada propiedad
    String x = prop.getProperty("es.caib.codiAplicacio.xxxx");
    String i = prop.getProperty("es.caib.codiAplicacio.yyyy");
    String z = prop.getProperty("es.caib.codiAplicacio.zzzz");

} catch (IOException ex) {
    ex.printStackTrace();
```



```
}
```

Las propiedades tendrán como prefijo *es.caib.codigoAplicacion*. Por ejemplo, el contenido del fichero *codigoAplicacion.properties* podría ser:

```
es.caib.codigoAplicacion.propiedad1=valor1  
es.caib.codigoAplicacion.propiedad2=valor2  
es.caib.codigoAplicacion.propiedad3=valor3
```

#### **4.5. Implantación de buenas prácticas en el código.**

El equipo de desarrollo de la aplicación deberá aplicar reglas de buenas prácticas generalmente reconocidas con el objetivo de reducir la aparición de errores en el código cuando el sistema se lleve a producción.

Algunas de estas buenas prácticas pasan por:

- Priorizar la legibilidad del código.
- Minimizar el acoplamiento (dependencia) entre módulos.
- Generar y ejecutar tests unitarios de todas aquellas clases que incluyan lógica de negocio.
- Evitar el uso de código redundante.
- Cerrar correctamente los ficheros y conexiones cuando ya no se necesiten.
- Eliminar variables, constantes, e importaciones de clases que no se utilicen.
- Establecer un correcto sistema de control de errores.
- Documentar y comentar adecuadamente el código.

Actualmente, la mayoría de entornos de desarrollo integrado, en inglés *Integrated Development Environment* (IDE), incluyen herramientas nativas o extensiones que facilitan la labor del desarrollador en este sentido.

#### **4.6. Restricciones adicionales**

Para todas aquellas funcionalidades no especificadas en este documento, se recomienda la utilización de estándares *Jakarta EE* en su versión 8 o, en su defecto, en su versión 7, evitando en la medida de lo posible soluciones particulares que solo funcionen sobre *JBoss*.



- Las aplicaciones deberán utilizar el juego de caracteres **UTF-8**:  
`<?xml version=... encoding="UTF-8" ?>`
- Los mensajes de depuración generados por la aplicación deberán aparecer en el fichero de log de la aplicación con categoría *DEBUG*, nunca *INFO* o *WARN*.
- Los *beans* serán preferentemente *stateless session beans*.
- Los *stateful session beans* deberán implementar adecuadamente los métodos *activate* y *passivate* al efecto de minimizar el consumo de memoria y recursos.
- Para todas aquellas funcionalidades no especificadas en este documento, se recomienda la utilización de los estándares Jakarta EE en su versión 8, evitando en lo posible soluciones particulares que solo funcionen sobre JBoss.



## 5. Seguridad

Todos los aspectos relativos a identificación y autorización de los usuarios a *servlets*, *REST*, controladores, *JSPs* o *EJBs* estarán gestionados de forma externa a las aplicaciones, desde el entorno de administración de la plataforma *Java EE*, por lo que no se debe codificar dentro de *servlets*, *REST*, controladores, *JSPs* o *EJBs* ninguna regla o criterio de autenticación. Sí pueden estar codificados dentro de la aplicación aspectos relativos a cómo se presenta el *interface* de usuario.

En caso de que la aplicación requiera restringir el acceso a los recursos mediante un usuario y contraseña deberán configurarse los elementos **security-constraint**, **login-config**, y **security-role** dentro del fichero **web.xml**.

### 5.1. Autenticación

A partir de la versión *EAP 7.2* de *JBoss*, la autenticación para acceder a los recursos protegidos de los módulos web se realiza mediante un mecanismo de autenticación centralizado: *RedHat Single Sign-On (Keycloak)*. El propio servidor de la aplicación (*JBoss*), al recibir una petición de acceso a un recurso que requiera autorización, se encargará de redirigir al usuario a la página de autenticación.

Dicho módulo de autenticación permitirá el control de accesos a las aplicaciones a nivel de *war* o módulo web. Por lo tanto, toda aplicación deberá estar separada en distintos módulos que requieran distintos mecanismos de autenticación. Éstos se configuran en *Keycloak* como *clients* y se indican en el fichero *standalone.xml* del *JBoss* como *resources*.

Para utilizar el nuevo módulo de autenticación es necesario instalar el adaptador de *RedHat Single Sign-On* sobre *JBoss*, para ello hay que descargar el adaptador de *RH-SSO 7.3* y aplicarlo a la instancia de *JBoss*:

```
$ cd $JBASS_HOME\bin
$ unzip rh-sso-7.3.0.GA-eap7-adapter.zip
$ ./jboss-cli.sh -file=adapter-install-offline.cli
```



Este adaptador instalará un módulo o subsistema en el fichero standalone.xml.

Los parámetros principales de configuración del subsistema de autenticación son los siguientes:

- **real-name** y **realm**: Nombre del dominio de confianza definido en Keycloak (normalmente pondremos «GOIB»).
- **auth-server-url**: Dirección del servidor de Keycloak. Generalmente será una dirección de un servidor local (el servidor de desarrollo de la DGEDDT está destinado por las aplicaciones corporativas). Para más información ver documento «Guía de configuración del entorno de desarrollo».
- **ssl\_required**: Para configurar si aceptamos conexiones SSL. Los posibles valores son: «ALL» (para todas las peticiones), «EXTERNAL» (por peticiones externas), o «NONE» (ninguna).
- **resource**: En los servidores de la DGEDDT hay disponibles tres recursos:
  - **goib-default**: permite al usuario autenticarse con certificado digital (cualquiera de los admitidos por @firma) o, si no dispone de certificado, mediante su usuario y contraseña del GOIB.
  - **goib-ws**: se trata del mecanismo de autenticación que tiene que aplicarse para proteger módulos que contengan servicios REST. Permite autenticación BASIC sin redirigir la petición en la web centralizada de autenticación.
  - **goib-cert**: solo habilita la autenticación con certificado, con lo cual, solo se podrá acceder a los contenidos protegidos del módulo, mediante el uso de un certificado válido.

A continuación se muestra un ejemplo de configuración para un módulo web de backoffice:



```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
  <realm name="GOIB">
    <auth-server-url>URL_DEL_KEYCLOAK</auth-server-url>
    <ssl-required>ALL</ssl-required>
  </realm>
  <secure-deployment name="projectbase-back.war">
    <realm>GOIB</realm>
    <resource>goib-default</resource>
    <use-resource-role-mappings>>false</use-resource-role-mappings>
    <public-client>true</public-client>
    <verify-token-audience>true</verify-token-audience>
  </secure-deployment>
</subsystem>
```

A continuación se muestra un ejemplo de configuración para un módulo **API REST**<sup>23</sup>:

```
<secure-deployment name="projectbase-api.war">
  <realm>GOIB</realm>
  <auth-server-url>URL_DEL_KEYCLOAK</auth-server-url>
  <resource>goib-ws</resource>
  <use-resource-role-mappings>true</use-resource-role-mappings>
  <bearer-only>true</bearer-only>
  <enable-basic-auth>true</enable-basic-auth>
  <ssl-required>ALL</ssl-required>
  <credential name="secret">CREDENTIAL_SECRET</credential>
</secure-deployment>
```

## 5.2. Control de acceso al módulo web

En el fichero **web.xml** de la aplicación se tienen que definir los roles de acceso permitidos (**security-role**). Los nombres de este roles tienen que seguir las normas estandarizadas de la DGEDDT. Para el caso de una aplicación con prefijo «APL» el nombre especificado al hashtag **role-name** tiene que seguir el formato «**APL\_XXXXXX**», donde «XXXXXX» tiene que ser un nombre simple y representativo.

Ejemplos de nombres de roles:

- APL\_CONSULTA

<sup>2</sup> Los servidores de la DGEDDT tienen activada la opción *Standard Flow Enabled del cliente goib-ws*.

<sup>3</sup> Los servicios REST requieren activar el parámetro «use-resource-role-mappings» y añadir un «credential name».



- APL\_USUARI
- APL\_ADMIN

Ejemplo de elemento *security-role*:

```
<security-role>  
<description> ... descripció ...</description>  
<role-name>PBE_ADMIN</role-name>  
</security-role>
```

En el mismo fichero, el elemento **security-constraint** se tendrá que utilizar en caso de tener que definir privilegios de acceso para una colección de recursos. Dentro de este elemento, tendrá que especificarse los siguientes parámetros:

- **auth-constraint/role-name**: Listado de roles que tendrán acceso a los recursos protegidos. Este roles se tendrán que definir previamente dentro del elemento *security-role*. Por ejemplo: PBE\_ADMIN.
- **login-config/auth-method**: Método de autenticación. Tendrá siempre el valor «KEYCLOAK»; independiente si se quiere usar autenticación con usuario / contraseña (BASIC/FORM) o con certificado (CLIENTE-CERT). De este modo, no será necesario modificar la aplicación si se quiere modificar el tipo de acceso puesto que este comportamiento está definido dentro del Keycloak y dentro de la configuración del fichero *standalone.xml* del JBoss.

A continuación se muestra un ejemplo de configuración del *web.xml* para restringir el acceso a todo el entorno del backoffice del Proyecto Base:



```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Tots els recursos</web-resource-name>
    <description>Tots els recursos</description>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>PBE_ADMIN</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>KEYCLOAK</auth-method>
  <realm-name>Govern dels Illes Balears</realm-name>
</login-config>
<security-role>
  <role-name>PBE_ADMIN</role-name>
</security-role>
```

### 5.3. Protección de EJBs

Es necesario proteger los EJBs de forma que ningún usuario o servicio anónimo pueda ejecutarlos, salvo que los EJBs tengan que ser públicos.

Si se requiere que el usuario tenga asignado uno o varios roles específicos se añadirá la anotación **@RolesAllowed({"ROL1", "ROL2",...})** a toda la clase o al método concreto en función del nivel de seguridad requerido.

Si se requiere que el usuario esté autenticado, pero no hace falta que tenga ningún rol concreto, se añadirá la anotación **@RolesAllowed("\*\*")**. Esto substituye al antiguo rol "tothom" utilizado en JBoss 5.

En el supuesto de que los EJBs tengan que ser públicos, se tendrá que especificar la anotación **@PermitAll**.

Ejemplo:

```
@RolesAllowed({"PBE_ADMIN"})
public class FooService implements FooServiceInterface {
  ...
}
```



#### **5.4. Dominios de seguridad**

No se tendrá que especificar ningún security-domain, ni explícitamente ni en los descriptores jboss-jar.xml, jboss-web.xml o jboss.xml. Si estos ficheros solo se usan para indicar el security-domain, estos se pueden eliminar directamente.

Así mismo, tampoco se tendrá que especificar ninguna referencia al security-domain mediante anotaciones en el código fuente (@SecurityDomain).



## 6. Servicios web y API REST

Las aplicaciones desarrolladas tienen que publicar servicios Web que permitan la integración otras aplicaciones o la consulta de datos.

**Se restringe el uso de servicios SOAP en favor de servicios REST.** Si se desean publicar servicios SOAP se tendrá que consultar con la DGEDDT y justificar su utilización.

La publicación de los servicios REST se hará mediante una API REST. Se podrán definir dos tipos d'API:

- **API interna.** Se publicarán los servicios REST de uso interno a la Administración de la Comunidad Autónoma de las Islas Baleares. Estos servicios REST permitirán la integración con otros sistemas propios de la CAIB y/o la consulta interna de los datos. Estos servicios REST no serán accesibles desde Internet. Las APIs REST internas podrán estar securizadas.
- **API externa.** Se publicarán los servicios REST de uso externo a la Administración de la Comunidad Autónoma de las Islas Baleares. Estos servicios REST serán accesibles desde Internet y permitirán la integración con sistemas externos a la CAIB y/o la consulta externa de los datos. Estas API REST podrán estar securizadas.

Remarcas para la definición del API REST (interna y/o externa):

- 1) Se tendrá que crear un submódulo **Maven** denominado **códigoAplicación-api-interna** y **códigoAplicación-api-externa** bajo el proyecto padre donde se situarán los recursos del API.
- 2) El API se tendrá que publicar siempre en el **contexto** **/códigoAplicaciónapi/interna/\*** o **/códigoAplicaciónapi/externa/\***, según el tipo de API.
- 3) Se tendrá que generar una documentación del API de forma automática mediante una herramienta tipo **Swagger**. Será opcional la utilización de una herramienta visual tipo Swagger-UI. La página de publicación tendrá que ser siempre **swagger.json** o **index.html**



dentro del contexto `/códigoAplicaciónapi/interna/` o `/códigoAplicaciónapi/externa/` según corresponda.

Ejemplo de documentación API publicada con *Swagger-UI*:

**foo** Show/Hide List Operations Expand Operations

**POST** `/foo/add/{value}` Permite dar de alta un elemento

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
value	<input type="text" value="(required)"/>	Valor de la entidad	path	string

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
200	OK		
500	Algo ha fallado en el servidor		

---

**GET** `/foo/list` Retorna lista de elementos

**Implementation Notes**  
Algo más?

**Response Messages**

HTTP Status Code	Reason	Response Model	Headers
200	OK		
500	Algo ha fallado en el servidor		

[ BASE URL: /proyectobase/api/services , API VERSION: 1.0.0 ]