Estàndards de desenvolupament d'aplicacions del GOIB.

Projecte Base

GOIB CONSELLERIA ADMINISTRACIONS PÚBLIQUES I MODERNITZACIÓ / DIRECCIÓ GENERAL MODERNITZACIÓ I ADMINISTRACIÓ DIGITAL

GOIB CONSELLERIA ADMINISTRACIONS PÚBLIQUES I MODERNITZACIÓ / FUNDACIÓ BALEAR INNOVACIÓ I TECNOLOGIA

Palma, gener de 2021

Índex

C/ de Sant Pere, 7
07012 Palma
Tel. 971 17 66 00
http://dgtic.caib.es

Historial de versions

Data	Versió	Descripció	Autor
03/12/20	1.0	Separació del document Jakarta EE i traducció al català	DGMAD
19/01/21	2.0	Afegir les principals qüestions a tenir en compte per tal migrar projectes ja existents	DGMAD, FBIT

C/ de Sant Pere, 7
07012 Palma
Tel. 971 17 66 00
http://dgtic.caib.es

1. INTRODUCCIÓ

Com es descriu en el document «Estàndard d'implantació d'aplicacions», és necessari adequar-se als estàndards de desenvolupament d'aplicacions per desenvolupar aplicacions pel GOIB. Per facilitar l'adequació a aquest estàndards, la DGMAD posa a disposició dels desenvolupadors **Projecte Base:** un projecte Jakarta EE d'exemple amb tota la tecnologia, frameworks, i estructura requerida.

Cal tenir en compte que Projecte Base és una guia pel desenvolupador; en cap cas és una plantilla d'obligat compliment en el desenvolupament d'aplicacions web.

En aquest document es descriurem les funcionalitats principals del Projecte Base, el procés per generar una aplicació basada en aquest, i les principals qüestions a tenir en compte per migrar projectes ja existents basats en els estàndards anteriors.

2. Projecte Base

Projecte Base és un **arquetip**[1] (*archetype*) MAVEN que permet automatitzar i parametritzar la creació de nous projectes Jakarta EE basats en els estàndards de desenvolupament del GOIB.

Projecte base proporciona:

- Exemple d'estructura de mòduls i submòduls MAVEN.
- Exemple d'empaquetament i estructura resultant esperada.
- Exemples de nomenclatura de: paquets, classes, atributs, mètodes, tests, etc.
- Exemple d'aplicació completa seguint l'especificació Java EE 8.
- Exemple de publicació de serveis REST amb JAX-RS .
- Exemple de publicació de documentació d'API REST amb Swagger i Swagger-UI.
- Clients d'integració amb les principals aplicacions d'administració electrònica[2] del GOIB.

1 Un arquetip és el patró exemplar del qual es deriven altres objectes, idees o conceptes.
2 Aquestes aplicacions estan descrites en el document «Catàleg de serveis i utilitats pels desenvolupadors» dels estàndards de desenvolupament.

C/ de Sant Pere, 7
07012 Palma
Tel. 971 17 66 00
http://dgtic.caib.es

3. Generació de nous projectes

El procés per la generació d'un projecte Jakarta EE mitjançant l'arquetip Maven del Projecte Base requereix tenir instal·lat **OpenJDK 11** i **Maven 3.6 o superior** (per més informació, consultar el document «Guia de configuració de l'entorn de desenvolupament local»).

3.1. Generació del projecte

Els passos a seguir són els següents:

1. Crear el fitxer de propietats MAVEN de l'usuari **[HOME]/.m2/settings.xml** (si no existeix) i afegir-li el següent contingut (bloc <profile> i <activeProfile>):

```
<settings xmlns="http://maven.apache.org/settings/1.0.0"
xmlns:xsi="http://www.w3.org/2001/xmlschema-instance"
        xsi:schemaLocation="http://maven.apache.org/settings/1.0.0 https://maven.apache.org/
       xsd/settings-1.0.0.xsd">
       <profiles>
          <profile>
           <id>repositorigovernib</id>
           <repositories>
            <repository>
              <id>github-governib-maven</id>
              <name>GitHub GovernIB Maven Repository</name>
              <url>https://governib.github.io/maven/maven/</url>
            </repository>
           </repositories>
          </profile>
       </profiles>
       <activeProfiles>
          <activeProfile>repositorigovernib</activeProfile>
       </activeProfiles>
</settings>
```

2. Des de la línia de comandes executar la comanda **mvn archetype:generate** amb els paràmetres i perfils descrits en l'apartat següent per generar el projecte. Aquesta comanda crearà el projecte dins d'un nou directori.

3.2. Descripció dels paràmetres i perfils

L'arquetip permet personalitzar les dades i mòduls del projecte a generar.

Per una banda, els paràmetres de configuració disponibles (que s'han d'indicar precedits de «-D») són:

- **archetypeVersion**. Permet especificar la versió de l'arquetip del Projecte Base. Es recomana utilitzar la darrera versió (la «1.0.8» en aquests moments).
- **package**. Jerarquia principal dels paquets (fins al tercer nivell). Segons els estàndards, totes les aplicacions hauran de tenir com a mínim quatre nivells: els dos primers nivells seran sempre «**es.caib**», el tercer nivell indicarà el nom de l'aplicació, i el quart nivell el mòdul (per exemple, «es.caib.projectebaseexemple»).
- **inversepackage**. Jerarquia de paquets inversa (per exemple, «projectebaseexemple.caib.es»).
- **packagepath**. Camí (path) dels paquets dins l'estructura de directoris (per exemple, «es/caib/projectebaseexemple»).
- **groupId**. Identificador del projecte MAVEN dins la resta de mòduls (per exemple, «es.caib.projectebaseexemple»).
- **artifactId**. Identificador de l'artefacte MAVEN (per exemple, «projectebaseexemple»).
- **version**. Versió del projecte generat (per exemple, «1.0.0»).
- **projectname**. Nom del projecte a generar (per exemple, «ProjecteBaseExemple»).
- **projectnameuppercase**. Nom del projecte en majúscules a generar (per exemple, «PROJECTEBASEEXEMPLE»).
- **prefix**. Prefix de tres caràcters designat per la DGMAD en el document d'inici de projecte (per exemple, «pbe»).
- **prefixuppercase**. Prefix de tres caràcters en majúscules (per exemple, «PBE»).

Per altra banda, l'arquetip permet especificar diferents **perfils** opcionals per la creació dels mòduls MAVEN del nou projecte:

- **perfilBack**. Generar mòdul de backoffice autenticat (frontal privat).
- **perfilFront**. Generar mòdul de frontoffice (frontal públic).
- **perfilApiExterna**. Generar mòdul de publicació d'API REST públic susceptible de ser publicat en Internet per dades obertes.[3]
- **perfilApiInterna**. Generar mòdul de publicació d'API REST privat per integracions amb aplicacions internes.
- **perfilApiFirmaSimple**. Generar mòdul d'integració amb l'API de firma simple (Portafib).
- **perfilArxiu**. Generar mòdul d'integració amb l'Arxiu digital.
- **perfilRegistre**. Generar mòdul d'integració amb el registre (REGWEB3).
- **perfilNotib**. Generar mòdul d'integració amb el servei de notificacions i comunicacions del GOIB (NOTIB).
- **perfilDir3Caib**. Generar mòdul d'integració amb el DIR3 del GOIB.
- **perfilDistribucio**. Generar mòdul d'integració com a backoffice de DISTRIBUCIÓ.
- **perfilSistra2**. Generar mòdul d'integració amb SISTRA2 mitjançant DISTRIBUCIÓ per obtenir les dades i documents dels tràmits.

A continuació es mostra un exemple de generació d'un projecte anomenat «ProjecteBaseExemple» amb tots els perfils disponibles. Per generar altre projecte s'ha de reemplaçar el text en negreta amb les dades del nou projecte (alternativament, es pot accedir a la documentació del Projecte Base i utilitzar el generador de projectes).

3 En compliment dels principis recollits per la «Llei 19/2013, de 9 de desembre, de transparència, accés a la informació pública i bon govern», és obligatori la publicació d'una API REST on es puguin consultar totes les dades susceptibles de ser reutilitzats per altres aplicacions o per ser publicats en formats oberts per a la seva possible reutilització per usuaris externs al GOIB.

```
set MAVEN_OPTS="-Dfile.encoding=UTF-8" && mvn org.apache.maven.plugins:maven-archetype-plugin:3.2.0:generate -B -DarchetypeGroupId=es.caib.projectebase -DarchetypeArtifactId=projectebase-archetype -DarchetypeVersion=1.0.8 -Dpackage=es.caib.projectebaseexemple -Dpackagepath=es/caib/projectebaseexemple -Dinversepackage=projectebaseexemple.caib.es -DgroupId=es.caib.projectebaseexemple -DartifactId=projectebaseexemple -Dversion=1.0.0 -Dprojectname=ProjecteBaseExemple -Dprojectnameuppercase=PROJECTEBASEEXEMPLE -Dprefix=pbe -Dprefixuppercase=PBE -DperfilBack=true -DperfilFront=true -DperfilApiInterna=true -DperfilApiExterna=true -DperfilApiFirmaSimple=true -DperfilArxiu=true -DperfilRegistre=true -DperfilNotib=true -DperfilDir3Caib=true -DperfilDistribucio=true -DperfilSistra2=true
```

Es recomana executar la comanda inicial **set MAVEN_OPTS="-Dfile.encoding=UTF-8"** per generar els fitxers amb codificació UTF8.

Important:

Un error molt comú consisteix en descarregar-se manualment el codi font del Projecte Base des del repositori de Github. No s'ha de confondre el codi font del Projecte Base amb l'ús d'aquest com a arquetip. En el nostre cas, només ens interessa la generació del nou projecte, per tant, **no cal descarregar-se el codi font del Projecte Base** sinó generar un projecte amb la comanda anterior que descarregarà l'arquetip com a dependència MAVEN.

3.3. Descripció dels directoris i fitxers generats

L'execució de l'arquetip amb tots els perfils activats genera els següents directoris i fitxers:

- **.gitignore**. Especifica els directoris i fitxers del projecte que no s'han de rastrejar dins d'un sistema de control de versions Git.
- **pom.xml**. Fitxer pom.xml del mòdul MAVEN pare.
- **doc**. Directori principal per la documentació del projecte (manual d'usuari, manual d'instal·lació, manual de migració,...).
- **codiAplicació-api-externa**. Definició dels serveis REST públics.
- **codiAplicació-api-interna**. Definició dels serveis REST privats.
- **codiAplicació-apifirmasimple**. Client d'integració amb l'API de Firma Simple (Portafib).

- **codiAplicació-arxiu**. Client d'integració amb l'Arxiu Digital.
- **codiAplicació-back**. Mòdul web pel backoffice.
- **codiAplicació-commons**. Classes i mètodes amb funcionalitat comú en tots els mòduls.
- **codiAplicació-dir3caib**. Client d'integració amb DIR3.
- **codiAplicació-distribucio**. Client d'integració amb Distribució.
- **codiAplicació-ear**. Mòdul per l'empaquetament de la resta de mòduls.
- **codiAplicació-ejb**. Serveis per accedir a la capa de persistència.
- **codiAplicació-front**. Mòdul web pel front-office basat en la plantilla per a aplicacions web públiques accessibles publicada en la web d'estàndards.
- **codiAplicació-notib**. Client d'integració amb Notib.
- **codiAplicació-persistence**. Capa de persistència de base de dades.
- **codiAplicació-registre**. Client d'integració amb Registre web.
- **codiAplicació-service**. Capa de lògica de negoci.
- **codiAplicació-sistra2**. Client d'integració amb SISTRA2.
- **README.md**. Document simple on s'han de descriure els requisits i configuracions per instal·lar el projecte des de zero.
- **scripts**. Directori que conté:
    - Els fitxers «**01_create_schema.sql**» i «**02_sample_data.sql**» amb els scripts sql a executar per crear els objectes i les dades dins d'una base de dades Oracle o PostgreSQL des de zero.
    - El fitxer «**drop_schema.sql**» a executar per esborrar els objectes de la base de dades.
    - El fitxer «**codiAplicació-ds.xml**» per definir el *datasource* de connexió amb el servidor de base de dades. S'ha de copiar dins del directori de desplegaments del JBoss.

- El fitxer «**keycloak-subsystem.xml**» que conté la configuració del submòdul Keycloak. S'ha de copiar el contingut dins del fitxer «standalone.xml» (o «standalone-full.xml») del JBoss,
- Els fitxers «**codiAplicació.properties**» i «**codiAplicació.system.properties**» que contenen les propietats internes de l'aplicació. Cal recordar que en el cas del primer fitxer la DGMAD en cap cas es farà càrrec d'afegir propietats o modificar els valors (s'haurà d'enviar a desplegament el fitxer sencer que ja incorpori els canvis respecte a l'anterior). S'han de crear les propietats de configuració «es.caib.codiAplicació.properties» i «es.caib.codiAplicacio.system.properties» que apuntin a la ruta on s'ubiquin els fitxers corresponents.

- **version-ruleset.xml**. Regles de noms de versions per emprar el «versions-maven-plugin».

3.4. Compilació del projecte generat

Una vegada generat el projecte, ja es pot començar a desenvolupar les funcionalitats concretes de l'aplicació per després compilar-lo.

Per compilar el projecte s'ha d'executar la comanda **mvn [opcions de compilació] clean install** des del directori pare del projecte o configurant el plugin de MAVEN dins de l'entorn de desenvolupament integrat (Eclipse, Intellij, Netbeans,..).

Els «profiles» disponibles per compilar (que s'han d'indicar precedits de «-P») són:

- **swagger-ui**. Inclou els HTMLs i recursos de swagger-ui (accessible des de /codiAplicació/api/). Swagger és un conjunt d'eines per a dissenyar, construir, documentar, i utilitzar serveis RESTful.
- **produccio**. Per activar els valors per enviar a producció (per exemple, s'estableix el valor per defecte dins JBoss per la propietat persistenceXml.hibernate.showSql).
- **authBasic**. Empra BASIC com a mètode d'autenticació dins els mòduls web, enlloc del valor per defecte, KEYCLOAK

- **testsIntegracio**. Executa els tests d'integració. Únicament cal tenir una JBoss instal·lat. No cal cap configuració addicional.
- **arq-jboss-managed**. Perfil activat per defecte. Si s'executen els tests d'integració arrancar el servidor de JBoss indicada amb la variable d'entorn JBOSS_HOME.
- **arq-jboss-remote**. Alternativa al perfil anterior. Si s'executen els tests d'integració emprar un JBoss ja arrancat, que estigui a «localhost:8080».

Si tot ha anat ve, es mostrarà un resum de la compilació amb el literal «SUCCESS» per a cada mòdul i es generarà el fitxer **ear.**

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for ProjecteBaseExemple 1.0.0:
[INFO]
[INFO] ProjecteBaseExemple ..... SUCCESS [  0.479 s]
[INFO] ProjecteBaseExemple - COMMONS ..... SUCCESS [  2.221 s]
[INFO] ProjecteBaseExemple - PERSISTENCE ..... SUCCESS [  0.931 s]
[INFO] ProjecteBaseExemple - EJB ..... SUCCESS [  0.601 s]
[INFO] ProjecteBaseExemple - BACK ..... SUCCESS [  2.322 s]
[INFO] ProjecteBaseExemple - FRONT ..... SUCCESS [  0.502 s]
[INFO] ProjecteBaseExemple - REST - API ..... SUCCESS [  1.160 s]
[INFO] projectebaseexemple-apifirmasimple ..... SUCCESS [  1.565 s]
[INFO] ProjecteBaseExemple - ARXIU ..... SUCCESS [  0.705 s]
[INFO] ProjecteBaseExemple - REGISTRE ..... SUCCESS [  0.423 s]
[INFO] ProjecteBaseExemple - NOTIB ..... SUCCESS [  0.602 s]
[INFO] ProjecteBaseExemple - DIR3CAIB ..... SUCCESS [  0.638 s]
[INFO] ProjecteBaseExemple - DISTRIBUCIO ..... SUCCESS [  6.675 s]
[INFO] ProjecteBaseExemple - EAR ..... SUCCESS [  0.795 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  19.287 s
[INFO] Finished at: 2020-10-09T11:32:30+02:00
[INFO] ------------------------------------------------------------------------
```

Una vegada compilat, s'ha de copiar el fitxer codiAplicació-ear/target/codiAplicació.ear dins del directori de desplegaments JBOSS_HOME\standalone\deployments.

Si està fixada la varialbe d'entorn «JBOSS_HOME» es pot desplegar automàticament, emprant la companda Maven «cargo:deploy».

Algunes consideracions a tenir en compte:

- El codi font generat per implementar les unitats organitzatives, els procediments, i els clients d'integració són només un exemple per mostrar la funcionalitat del projecte. És probable que aquest codi no sigui necessari dins l'aplicació i s'hagui d'esborrar.
- El projecte generat està orientat a un entorn de desenvolupament; per la qual cosa s'hauran de revisar les propietats de l'aplicació en la sol·licitud de desplegament a preproducció i producció. En concret, el valor de la propietat **hibernate.show_sql** definida dins del fitxer **codiAplicació-persistence\src\main\resources\META-INF\ persistence.xml** haurà de tenir sempre valor **false** en aquests entorns. Dins la confiugració de Maven s'empra el perfil «produccio» ja esmentat per automatitzar l'ús de valors adequats per desenvolupament o producció.
- En la sol·licitud de desplegament s'ha de demanar la creació dels usuaris d'integració definits en els fitxers de propietats. En el document «Catàleg de serveis i eines per a desenvolupadors» apareix un llistat amb el format dels noms d'usuaris i els rols que cal demanar per a cada servei.

C/ de Sant Pere, 7
07012 Palma
Tel. 971 17 66 00
http://dgtic.caib.es

4. Migració de projectes ja existents

L'arquetip genera un projecte nou adaptat als actuals estàndards de desenvolupament del GOIB.

Per tal d'adaptar un projecte ja existent basat en la plataforma definida en els anteriors estàndards (JBoss EAP 5.2 / Java 7) als actuals estàndards (JBoss EAP 7.2 / Java 11) es recomana generar un projecte nou amb l'arquetip, copiar-hi les classes i la resta del codi del projecte antic, i realitzar els canvis oportuns. A continuació es descriuen les principals qüestions a tenir en compte durant aquest procés.

4.1. Canvis instal·lació i configuració d'aplicacions

4.1.1. Fitxers de configuració

A la versió anterior dels estàndards, les propietats de configuració s'establien mitjançant un fitxer «-service.xml» del qual es feia «deploy» al servidor d'aplicacions i aquesta definia propietats de sistema que podien ser consultades mitjançant cridades a «System.getProperty».

En la versió actual dels estàndards aquest mètode ha canviat. Ara és necessari establir dues propietats de sistema que apuntaran a sengles fitxers amb la definició de les propietats que haurà de carregar l'aplicació. Es pot veure amb més detall aquest sistema així com un exemple a la secció 4.5 del document «Estàndards Jakarta EE».

Una conseqüència important d'aquest canvi és que les propietats deixaran d'estar accessibles mitjançant una cridada a «**System.getProperty**». Per tant, s'hauran de canviar aquestes cridades, i en cas de fer ús de frameworks o llibreries que obtenguin propietats de configuració mitjançant aquest sistema caldrà cercar alternatives per subministrar les propietats de configuració.

4.1.2. Microprofile config

El servidor JBoss EAP 7.2 suporta l'especificació Microprofile Config 1.3 que defineix un sistema per registrar orígens de propietats de configuració i, d'altra banda, una API i unes anotacions que permeten obtenir aquestes.

Dins el mòdul «-common» de ProjecteBase hi ha una implementació d'origen de propietats de configuració complint els estàndards GOIB (és la classe «PropertyFileConfigSource» que es registre emprant el mecanisme estàndard «service loader»).

Per tal d'emprar després les propietats de configuració es pot fer «programàticament» emprant l'API de MicroProfile Config:

```
Config config = ConfigProvider.getConfig();
String valor = config.getValue("es.caib.projectebase.valor", String.class);
```

O emprant CDI, injectant directament el valor:

```
@Inject
@ConfigProperty(name = "es.caib.projectebase.valor")
private String valor;
```

4.1.3. Datasources

La nova versió de JBoss EAP 7.2 segueix suportant el «deploy» de fitxers «-ds.xml» per configurar els datasources, però el format dels fitxers ha canviat. A dins els projectes generats amb ProjecteBase, a la carpeta «scripts/datasources» hi ha exemples dels datasources amb el nou format.

4.1.4. Autenticació

Les aplicacions desenvolupades per JBoss EAP 5.2 amb els estàndards anteriors del GOIB s'integraven amb el sistema d'autenticació SEYCON. Ara a les aplicacions desenvolupades per JBoss EAP 7.2 s'empra el sistema **KEYCLOAK**.

El document «Guia de configuració de l'entorn desenvolupament» detalla les passes a seguir per a configurar les aplicacions per KEYCLOAK.

Per migrar les aplicacions cal tenir en compte que:

- Ja no cal definir el fitxer «jboss-web.xml» als mòduls WAR, ni «jboss.xml» als móduls EJB per definir el «security-domain».
- Tampoc s'han d'anotar els EJBs amb l'anotació «@SecurityDomain("seycon")» (i per tant es poden eliminar les dependències al projecte cap a «jboss-ejb3-ext-api»

- Al web.xml es recomana fer servir «KEYCLOAK» com a «auth-method»
- Si s'han d'obtenir dades de l'usuari cal fer servir el «KeycloakPrincipal». Dins ProjecteBase a la classe «KeycloakUserInfoFactory» hi ha un exemple de com obtenir les dades de l'usuari del sistema d'autenticació (nom, cognoms, correu electrònic i nif)

4.1.5. Classloader

En principi les aplicacions desplegades sobre JBoss EAP 7.2 empren un sistema més modular per carregar les dependències, de manera que s'eviten millor els conflictes amb altres aplicacions i amb els mòduls que proporciona el servidor d'aplicacions.

El servidor afegeix dins el classpath de l'aplicació una sèrie de mòduls de forma implícita, unes estan disponibles per totes les aplicacions, i d'altres s'activen automàticament en funció de les característiques d'aquesta.

En aquest enllaç es pot veure els mòduls que s'afegiran implícitament sempre, els que s'afegiran de forma opcional, i les condicions en les que s'afegiran. En aquest altre enllaç es pot veure els mòduls que estan disponibles dins el servidor i la seva classificació.

Per exemple, en el cas de sistema de logging, JBoss posa automàticament els mòduls de «commons-logging», «log4», «sl4j» i «jboss-logging» dins el classpath, per tant caldrà tenir en compte que si tenim aquestes dependències dins el nostre projecte, sempre les posarem a «provided» dins Maven i no s'incloguin dins el WEB-INF/lib dels WARs, ni dins els EAR.

D'altra banda, per assegurar que la versió d'una llibreria amb la que compilam es correspon amb la mateixa versió que es troba dins el JBoss podem emprar els BOM específics de JBoss dins Maven. Dins el «pom.xml» dels projectes generats amb ProjecteBase ja venen configurats d'aquesta manera 8 (veure les dependències «org.jboss.bom:eap-runtime-artifacts» i «org.jboss.bom:jboss-eap-javaee8»).

Si sorgeixen conflictes respecte les llibreries que necessita incorporar la nostra aplicació i llibreries que el JBoss afegeixi automàticament al

classpath perquè es tracti de versions diferents, caldrà emprar el fitxer jboss-deployment-structure.xml dins el directori META-INF del EAR (Els projectes generats amb ProjecteBase ja incorporen aquest fitxer).

Dins aquest es poden excloure mòduls del JBoss perquè les versions que incorpora l'aplicació tenguin preferència. En aquest enllaç es pot trobar la documentació completa del sistema de Classloading the JBoss EAP 7.2 que detalla com fer ús d'aquest mecanisme.

4.1.6. Canvis a la configuració Maven dels projectes

El principal canvi serà l'actualització de dependències, tenint en compte el que s'ha explicat al punt anterior 4.1.5 Classloader, així com al punt 4.2.9 Mòduls i APIs eliminades. Al marge d'això caldrà tenir en compte el següent:

- Es recomana emprar les darreres versions dels plugins maven, definint-los dins l'apartat «pluginManagement» del pom.xml arrel.
- El sistema per especificar la versió del codi font i del bytecode generat per la compilació que es podien establir mitjançant <source> i <target> dins el «maven-compiler-plugin», o amb les propietats «maven.compiler.source» i «maven.compiler.target», emprant el valor «1.7» ara (a partir de Java 9 i versió del plugin 3.6), cal especificar-ho amb <release>, o la propietat «maven.compiler.release», amb el valor «11».
- En els mòduls que contenguin EJB, al «maven-ejb-plugin» emprarem «3.2» dins <ejbVersion>
- En el mòdul EAR, al «maven-ear-plugin», especificarem la versió de Java EE amb el valor «8» a dins <version>
- En els mòduls que produeixin WAR, el «maven-war-plugin», a partir de la versió 3.1, el paràmetre <failOnMissingWebXml> té valor per defecte «false». Si volem que el build falli quan no es troba el descriptor web.xml caldrà fixar el valor a «true»

4.2. Canvis de Java 7 a Java 11

Les diferents versions de Java mantenien la compatibilitat cap enrera, per tant el codi que compila amb Java 7 compilarà també amb Java 11. Sí que caldrà tenir en compte que qualcunes de les APIs que formaven part de la màquina virtual ara són dependència externes, tal com es detalla al punt 4.2.9 Mòduls i APIs eliminades.

A continuació es detallen les principals novetats que a nivell de sintaxis i APIs s'han incorporat al llenguatge.

4.2.1. Expressions lambda

Des de Java 8 s'incorporen al llenguatge les expressions lambda, que permeten la definició de classes anònimes de forma molt més compacte.

Es poden emprar sempre que es tracti de crear classes anònimes de interfícies que només defineixen un mètode, i segueixen el format

(llista paràmetres separats per comes) → expressió

ó

(llista paràmetres separats per comes) → { bloc de codi}

Per exemple, el següent codi:

```
Thread thread = new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("Soc un thread nou");
    }
});
```

Atès que la interfície Runnable té un sòl mètode, sense cap paràmetre, es pot expressar de forma molt més compacte així:

```
Thread thread = new Thread(() -> System.out.println("Soc un thread nou"));
```

En el cas de voler crear un «Comparator», per exemple, el següent codi que compararia instàncies de «UnitatOrganicaDTO» en funció del seu codiDir3:

```
Comparator<UnitatOrganicaDTO> comparator = new
Comparator<UnitatOrganicaDTO>() {
    @Override
    public int compare(UnitatOrganicaDTO uo1, UnitatOrganicaDTO uo2) {
        return uo1.getCodiDir3().compareTo(uo2.getCodiDir3());
```

```
        }
    };
```

Es podria simplificar així:

```
Comparator<UnitatOrganicaDTO> comparator =
        (uo1, uo2) -> uo1.getCodiDir3().compareTo(uo2.getCodiDir3());
```

A part de poder expressar de forma molt més compacte la implementació de classes anònimes amb interfícies com «Runnable» o «Comparator» que tenen un sòl mètode, les expressions lambda formen part d'un canvi molt més profund, perquè a partir de Java 8 també s'incorporen tot un seguit de interfícies que com les anteriors, només requereixen de la implementació d'un mètode, anomenades «funcionals», i s'incorporen també a l'API utilitats que fan ús d'aquestes interfícies funcionals.

Per exemple, el cas del «Comparador» el podem simplificar més encara. El mètode d'utilitat «Comparator.comparing» ens retorna un «Comparator». Per paràmetre li hem de passar una implementació de la interfície funcional «Function», un interfície amb un sòl mètode «R apply(T t);», és dir, rep un objecte i en retorna un altre. El mètode d'utilitat «comparing» crearà un comparador que aplica aquesta funció a cadascun dels paràmetres i compara el resultat.

Emprant l'expressió « (uo) → uo.getCodiDir3()» tendríem una implementació d'aquesta funció que «extreu» el camp que volem comparar, per tant podríem simplificar el codi així:

```
Comparator<UnitatOrganicaDTO> comparator = Comparator.comparing(uo ->
uo.getCodiDir3());
```

(quan la llista de paràmetres és un sòl paràmetre es poden ometre els parèntesis, per tant podem posar «uo →» enlloc de «(uo) →»).

El codi anterior crea un comparador que rebrà dos objectes «UnitatOrganicaDTO», per cadascun cridarà el seu mètode «getCodiDir3()» i comparará aquests valors.

En els següents enllaços hi trobareu tutorials molt més complets d'expressions lambda:

- https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
- http://tutorials.jenkov.com/java/lambda-expressions.html

4.2.2. Referències a mètodes

Les referències a mètodes són un cas particular d'expressions lambda, quan l'expressió és limita a cridar un mètode.

En el darrer exemple del punt anterior, l'expressió lambda «uo → uo.getCodiDir()» que es limita a cridar el mètode getCodiDir3 sobre el paràmetre, es pot simplificar amb la referència al mètode que s'expressaria així: «UnitatOrganicaDTO::getCodiDir3», per tant podríem reescriure el codi així:

```
Comparator<UnitatOrganicaDTO> comparator =
Comparator.comparing(UnitatOrganicaDTO::getCodiDir3);
```

Les referències a mètodes expressades d'aquesta manera poden fer referència a un mètode «static» que rep els paràmetres, o a un mètode d'instància invocat sobre el paràmetre, en funció del context.

En els següents enllaços hi trobareu una descripció més completa de com emprar les referències a mètodes:

- https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html
- http://tutorials.jenkov.com/java/lambda-expressions.html#method-references-as-lambdas

4.2.3. Streams

Com s'avançava en l'apartat dedicat als lambda, dins l'API de Java 8 s'incorporen tot un seguit de noves classes i mètodes que fan ús extensiu de les interfícies funcionals. La principal d'aquestes APIs són els «Streams».

Els «Streams» són seqüències d'elements, en certa manera serien com una altre versió de les «Collection». Per exemplificar-ho d'alguna manera, si els «Collection» estan centrats en la representació dels elements (llistes, conjunts), els Streams es centren en el seu procés.

Els streams permeten manipular de forma fàcil seqüències d'elements, filtrar-los, mapejar-los a altres elements, ordenar-los, i agrupar-los de forma declarativa.

A mode d'exemple, suposem que tenim una llista de «UnitatOrganicaDTO», i volem filtrar els que tenen el camp «estat» a «ACTIU», ordenar-los per data creació, i obtenir una llista dels seus noms. Ho faríem amb el següent codi:

```
List<String> resultat = llista.stream()
                .filter(uo -> uo.getEstat().equals(EstatPublicacio.ACTIU))
                .sorted(Comparator.comparing(UnitatOrganicaDTO::getDataCreacio))
                .map(UnitatOrganicaDTO::getNom)
                .collect(Collectors.toList());
```

En primer lloc, les classes del API de Collections incorporen un mètode «stream» per transformar-les en streams. El mètode «filter» rep com a paràmetre una interfície «funcional» de tipus «Predicate» amb un sòl mètode que rep un objecte i retorna un booleà, que implementam amb l'expressió lambda «uo -> uo.getEstat().equals(EstatPublicacio.ACTIU)» de manera que només els elements de la seqüència que tenen el camp Estat amb el valor «ACTIU» compliran el filtre.

Mitjançant el mètode «sorted» podem passar un «Comparator» com el que hem vist a les seccions anteriors, que s'aplicarà al a seqüència d'elements per ordenar-los, en aquest cas comparant el camp «getDataCreacio».

El mètode «map» rep per paràmetre una «Function» que transforma els elements amb Strings aplicant el mètode «getNom».

I finalment amb «collect» agrupam els resultats amb un «Collector». En aquest cas el collector emprat el que fa és simplement retornar una llista.

En els següents enllaços hi trobareu varis tutorials per l'ús d'aquesta nova API:

- https://www.oracle.com/technical-resources/articles/java/ma14-java-se-8-streams.html
- http://tutorials.jenkov.com/java-functional-programming/streams.html

4.2.4. Optionals

Normalment, quan un mètode retorna un objecte, i en determinades circumstàncies aquest no existeix, fins ara tenia dues opcions, o bé llançar una excepció o bé retornar «null».

L'API de Java 8 introdueix un nou recurs per tractar aquest tipus de situacions quan volem expressar amb el resultat d'un mètode aquesta possible presència o no de resultat, seria la classe «Optional», que serveix per encapsular un resultat que pot ser «null» obligant al client de l'API a tenir en compte aquesta possibilitat, promoguent una forma de programar més segura que eviti les «NullPointerException».

Per exemple, enlloc de tenir un mètode del tipus:

```
UnitatOrganicaDTO findById(Long id);
```

Que pot tornar una instància de l'objecte o «null» si aquest no existeix, retornarem:

```
Optional<UnitatOrganicaDTO> findById(Long id);
```

De manera que obligam al client del mètode a contemplar la possibilitat de que el resultat és «null»:

```
Optional<UnitatOrganicaDTO> optional = unitatOrganicaServiceFacade.findById(id);
if (optinal.isPresent()) {
    UnitatOrganicaDTO uo = optional.get();
}
```

La classe «Optional» disposa de multitud de mètodes que s'han anat ampliat a les versions de Java 9, 10 i 11 per tractar les diferents casuístiques.

En els següents articles podeu trobar exemples de bones pràctiques de com emprar aquesta nova funcionalitat.

- https://www.oracle.com/technical-resources/articles/java/java8-optional.html
- https://www.baeldung.com/java-optional

4.2.5. Mètodes default i static a les interfícies

A partir de Java 8 es poden afegir mètodes a les interfícies definint una implementació per defecte. Això permet ampliar una interfície sense afectar a les classes que l'implementen, que seran siguent compatibles amb la nova versió de la interfície. També es permet la definició de mètodes «static» dins les interfícies.

En el següent enllaç s'explica aquesta nova funcionalitat:

- https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html

4.2.6. Declaració de variables locals amb «var»

A partir de Java 10, quan es declaren variables locals dins un mètode es pot ometre el tipus de la variable indicant la paraula «var».

Per exemple, enlloc de:

```
UnitatOrganicaDTO unitat = new UnitatOrganicaDTO();
```

Podem posar:

```
var unitat = new UnitatOrganicaDTO();
```

Podeu trobar més detalls i exemples de la nova paraula clau «var» en aquest enllaç:

- https://developers.redhat.com/blog/2018/05/25/simplify-local-variable-type-definition-using-the-java-10-var-keyword/

4.2.7. Nova API Java Time

Java 8 incorpora una nova API per gestionar dates i temps, al package «java.time» que corregeix els problemes amb els actuals java.util.Date i incorpora millores substancials.

Per exemple, si fins ara s'emprava un java.util.Date per representar tant una data (dia més any) com un instant concret (data + hora minut segon...), ara aquests tenen representacions específiques (LocalDate i LocalDateTime).

Tant els components actuals de JSF/Primefaces com els de JPA/Hibernate soporten aquesta API nova, fet que facilita la migració del model per emprarl a nova API.

En els següents enllaços podeu trobar més informació sobre aquesta nova API:

- https://www.oracle.com/technical-resources/articles/java/jf14-date-time.html
- https://www.baeldung.com/java-8-date-time-intro

4.2.8. Resource Bundle i UTF-8

Fins a Java 9, els fitxers d'etiqutes carregats com a «ResourceBundle» es llegien com a ISO-8859-1. A partir de Java 9, els fitxers d'etiquetes es llegeixen primer per defecte amb UTF-8, i si es produeix un error en la lectura, com a ISO-8859-1.

Per tant, si feiem ús de qualque recurs, com ara sobreescriure la classe ResourceBundle.Control per a poder tenir els fitxers d'etiquetes amb UTF-8, podrem eliminar aquest «workaround» i emprar els fitxers d'etiquetes directament.

4.2.9. Mòduls i APIs eliminades

A partir de Java 9 s'incorpora un canvi importat a nivell d'estructura de la màquina virtual, i és que aquesta és modularitza més. És proveeixen d'eines per la definició de mòduls, que vendria a ser una agrupació de «packages» que permet definir un nivell addicional de visibilitat per declarar per exemple les APIs que ofereix una llibreria, i eines per distribuir aplicacions amb la màquina virtual incorporant només els mòduls necessaris, disminuint per tant el pes d'aquestes.

Aquesta nova modularització en principi no té un efecte directe sobre les aplicacions «Java EE» perquè les especificacions que recullen els formats WAR i EAR per la generació d'aplicacions encara no han incorporat aquests conceptes.

Si que cal tenir en compte però que dins aquests procés de modularització, qualcuns dels mòduls resultants s'han eliminat de dins la distribució de la màquina virtual i per tant ara s'han convertit en dependències externes. Cal tenir en compte a l'hora de compilar les aplicacions ja que harem de definir aquestes dependències dins el projecte Maven (tot i que emprant

l'scope «provided» atès que aquestes dependències si que són presents dins el servidor d'aplicacions)

És el cas dels mòduls JAXB, JAX-WS, JAF i JTA . Tot i això, els projectes generats amb ProjecteBase, mitjançant la dependència «org.jboss.spec:jboss-javaee-8.0» ja incorporen aquests mòduls.

Podeu llegir més sobre el nou sistema de mòduls als següents enllaços:

- https://www.oracle.com/corporate/features/understanding-java-9-modules.html
- http://tutorials.jenkov.com/java/modules.html

4.2.10. Altres canvis

A part de les APIs mencionades, des de Java 7 fins a Java 11 s'han incorporat múltiples mètodes d'utilitat a les classes existents, en molts de casos relacionats amb les noves APIs de Streams i interfícies funcionals, facilitant l'ús d'expressions lambda, així com classes per a la programació reactiva.

Amb la següent eina es poden explorar els canvis introduïts en cada versió de Java, i comparar-los amb qualsevol de les versions anteriors: https://foojay.io/almanac/jdk-11/

4.3. Canvis Java EE 5 a Java EE 8

La versió de JBoss EAP 5.2 estava certificada com a servidor Java EE 5, mentre que el JBoss EAP 7.2 està certificat com a Java EE 8.

En el següent enllaç es pot veure i comparar la relació d'APIs suportades per les diferents versions de JBoss: https://access.redhat.com/articles/113373

4.3.1. Servlets

Les diferències entre l'especificació Servlet 2.5 present a Java EE 5, i l'especificació Servlet 4.0 són considerables, si bé l'especificació estableix que es manté la compatibilitat cap enrera.

Per adaptar el fitxer descriptor web.xml a la nova versió de l'especificació caldrà emprar:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        version="4.0">
...
</web-app>
```

A continuació es detallen les principals noves funcionalitats. L'especificació completa es pot descarregar aquí: https://download.oracle.com/otndocs/jcp/servlet-4-final-spec/index.html

CONFIGURACIÓ AMB ANOTACIONS I PROGRAMÀTICA

Tant els Servlets com els Filters com els Listeners es poden configurar mitjançant anotacions, sense necessitat d'especificar-los al fitxer web.xml amb les anotacions @WebServlet, @WebFilter, o @WebListener (veure: https://javaee.github.io/javaee-spec/javadocs/javax/servlet/annotation/package-summary.html).

De fet, a conseqüència de l'anterior, el fitxer web.xml ja no és obligatori.

Així mateix s'ha afegit tota una nou seguit de mètodes, principalment a la classe ServletContext que permet crear i registrar servlets, filters o listeners de forma programàtica, és a dir es pot afegir un nou servlet i mapejar-lo a una URL determinada «en calent».

SUPORT PER PUJADA DE FITXERS

L'especificació suporta directament la pujada de fitxes des de formularis, per tant, ja no calen llibreries externs per implementar aquesta funcionalitat.

Per configurar-la (a part d'emprar enctype="multipart/form-data" dins el formulari HTML), cal o bé emprar @MultipartConfig al Servlet que rebi el formulari si empram anotacions, o bé dins el web.xml emprar l'element <multipart-config>.

Veure: https://javaee.github.io/tutorial/servlets011.html

CODIFICACIÓ DE CARÀCTERS

Per motius de compatibilitat cap enrere, l'especificació Servlet segueix emprant ISO-8859-1 com a codificació de caràcters per defecte quan aquesta no s'especifica.

Això feia que habitualment s'empràs un Filter, com el «CharacterEncodingFilter» de Spring o similar, per fixar la codificació de l'objecte «request» abans de llegir la petició.

La nova especificació permet especificar directament la codificació a emprar per defecte a les peticions i a les respostes amb els elements <request-character-encoding> i <response-character-encoding>, fent innecessaris per tant aquests tipus de filtres.

D'altra banda, en el cas de la codificació de la URL (i per tant els paràmetres passats com a GET), es clarifica que s'ha de processar com a UTF-8. Cal dir que aquest sí és canvi respecte el comportament del servidor JBoss EAP 5.2 i concretament el connector Tomcat que duia incorporat que per defecte llegia les URLs i per tant els paràmetres GET amb ISO-8859-1 si no es configurava d'altra manera (veure https://tomcat.apache.org/tomcat-5.5-doc/config/http.html, atribut URIEncoding)

CONFIGURACIÓ DEL MANTENIMENT DE SESSIÓ

La configuració de les cookies per mantenir sessió i l'URL rewriting emprat a la versió 2.5 de l'especificació era molt limitat, i no permetia la seva adaptació als requisits actuals de seguretat.

L'element del web.xml <session-config> s'ha ampliat amb aquest aspecte, i permet especificar un o varis <tracking-mode>, de manera que si no especificam URL es deshabilitar URL rewriting, i en el cas de les cookies podem especificar, entre d'altres, <http-only> per evitar que el Javascript de les pàgines pugui accedir a les cookies.

ROL «**»

En les restriccions de seguretat definides al web.xml (o ara també amb anotacions) es pot emprar el nom de rol «**» per indicar que és necessari estar autenticat per accedir a un recurs, però que no es requereix cap rol concret.

REQUESTS ASÍNCRONS I NON-BLOCKING I/O

Aquestes dues noves funcionalitats permeten la creació d'aplicacions molt més escalables.

Posem per exemple que un Servlet ha d'accedir a un WebService extern per completar la petició, i aquest servei extern pot tardar molt de temps a respondre. Si arriben moltes peticions al servlet es poden exahurir el nombre de threads del servidor dedicats a servir peticions HTTP. Emprant requets asíncrons, el thread que serveix la petició pot tornar tot d'una i es delega la finalització de la petició quan ha acabat la cridada al web service a un altre thread. Veure https://javaee.github.io/tutorial/servlets012.html

D'altra banda, el non-blocking i/o permet que quan hi ha peticions o respostes molt grosses, i l'amplada de banda permet que el servidor escrigui o llegeixi molt més aviat del que es poden enviar les dades, aquest no es quedi bloquejat i es pugin atendre altres peticions. Veure https://javaee.github.io/tutorial/servlets013.html

4.3.2. EJBs

En el cas de l'especificació EJB, la versió present a JBoss EAP 5.2 era la 3.0 mentre que la del servidor JBoss EAP 7.2 és la 3.2.

Per adaptar el fitxer descriptor ejb-jar.xml a la nova versió de l'especificació caldrà emprar:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd"
         version="3.2">
...
</ejb-jar>
```

A continuació es detallen de forma no exhaustiva les principals novetats. L'especificació completa es pot descarregar aquí: https://download.oracle.com/otndocs/jcp/ejb-3_2-fr-spec/index.html

Cal tenir en compte que JPA ara és una especificació independent i es detalla a un altre apartat.

NOU MODEL DE NOMS JNDI

El sistema per establir els noms JNDI dels components EJB ha canviat considerablement. En cas de que l'aplicació no faci ús de la injecció directe mitjançant l'anotació @EJB i requreixi els noms JNDI caldrà tenir en compte que s'han d'adaptar els noms al nou model.

Veure https://javaee.github.io/tutorial/ejb-intro004.html#GIRGN

MÈTODES ASÍNCRONS

Es poden definir mètodes EJB asíncrons amb l'anotació @Asynchronous. Aquests mètodes retornen un objecte del tipus Future<X>, que permet esperar i accedir al resultat de l'execució.

Veure: https://javaee.github.io/tutorial/ejb-async001.html

SINGLETONS

Es poden definir EJBs «singleton» amb l'anotació @Singleton. Aquests EJBs per defecte funcionen amb control de concurrència estricte, de manera que les execucions de mètodes es serialitzen.

El control de concurrència es pot relaxar amb l'anotació @Lock per indicar mètodes que es poden executar en paral·lel (LockType.READ) o mètodes que requereixen executar-se sense concurrencia (LockType.WRITE).

Els EJBs Singleton també és poden inicialitzar a l'inici de l'aplicació amb l'anotació @Startup, que provocara la cridada del mètode @PostConstruct del mateix. Per tant són un emplaçament idoni per tasques d'inicialització de l'aplicació a la capa EJB, que fins ara només és podien realitzar a la capa web mitjançant la inicialització de servlets o de ServletContextListener.

Veure: https://javaee.github.io/tutorial/ejb-basicexamples003.html

MILLORES AMB LA PROGRAMACIÓ DE TASQUES (TIMERS)

En general s'ha millorat les possibilitats tant amb anotacions com programatiques dels timers. Per exemple, amb l'ús d'expressions semblants a CRON per programar timers (amb l'anotació @Schedule o els mètodes de TimerService).

També s'afegeix la possibilitat d'establir si els timers són persistents o no (amb JBoss EAP 5.2 els timers sempre eren persistents, i per tant si hi havia timers programats aquests s'executaven a l'inici del servidor d'aplicacions si s'havia produït un timeout durant el tems que el servidor estava aturat)

ROL «»**

Com en el cas dels Servlets, s'afegeix la possibilitat d'emprar el rol «**» amb l'anotació @RolesAllowed (o al descriptor ejb-jar.xml) que permet

establir que un mètode necessita que l'usuari estigui autenticat però no requereix cap rol concret.

4.3.3. JPA

JBoss EAP 5.2 soportava la Java Persistence API 1.0, que formava part de l'especificació EJB 3.0. Ara aquesta és una especificació de primer nivell, i JBoss EAP 7.2 suporta la versió 2.2.

Per adaptar el fitxer descriptor persistence.xml a la nova versió de l'especificació caldrà emprar:

```
<?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>
```

A continuació es detallen de forma no exhaustiva les principals novetats. L'especificació completa es pot descarregar aquí: https://download.oracle.com/otndocs/jcp/persistence-2_2-mrel-eval-spec/index.html

- Mapping: moltes més opcions a l'hora de mapejar col·leccions de primitives, de «embedded objects», llistes ordenades, «maps», taules de join per relacions N:M, identificadors derivats. Suport pel mapeig de les noves classes de l'API Java Time.
- Millores al llenguatge JPQL per suportar els nous mappings, a més de noves funcions (NULLIF, COALESCE), sentències CASE,possibilitat de cridar qualsevol funció definida a la base de dades (FUNCTION), possibilitat de fer «downcasts» amb TREAT...
- Noves TypeQuery per fer no haver de fer «casts» amb els resultats.
- Nova API Criteria i metamodel, per poder construir consultes i sentències DML programàticament de forma tipificada.
- Possibilitat de realitzar «locks» de base de dades.

- Integració amb la nova API Bean Validation per establir validacions sobre els valors dels camps.
- «Cache» de segon nivell.
- Possibilitat de realitzar cridades a «procedures» de base de dades.
- AttributeConverter: permeten convertir valors d'objectes a valors de base de dades.
- Més utilitats per mapejar el resultats de queries natives
- Introducció dels EntityGraphs que permeten controlar els atributs que es carreguen per una consulta.
- Eines per la generació automàtica de l'schema de base de dades.

En els següents articles trobareu una relació més completa de les funcionalitats afegides a les versions 2.0, 2.1 i 2.2:

- JPA 2.0: https://dzone.com/refcardz/whats-new-jpa-20?chapter=1
- JPA 2.1: https://thorben-janssen.com/jpa-21-overview/
- JPA 2.2: https://thorben-janssen.com/whats-new-in-jpa-2-2/

4.3.4. JSF

JBoss EAP 5.2 suportava l'especificació JSF 1.2. Tot i això, l'anterior versió de «ProjecteBase» per a JBoss EAP 5.2 incorporava la implementació de JSF 2.1, incorporant les llibreries de l'API de de la implementació dins els mòduls WAR. El servidor JBoss EAP 7.2 suporta la versió 2.3. Per tant, no caldrà incloure dins els projectes les llibreries jsf-impl i jsf-api de com.sun.faces per passar de la versió JSF 2.1 a la versió 2.3.

Als següents punts es detallaran de forma no exhaustiva les principals novetats. L'especificació completa es pot descarregar aquí:

- https://download.oracle.com/otndocs/jcp/jsf-2_3-final-spec/index.html

Per activar qualcunes de les novetats caldrà que s'actualitzi el descriptor faces-config.xml a la nova versió amb:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<faces-config
        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_3.xsd"
        version="2.3">
...
</faces-config>
```

Així mateix, caldrà incloure una classe que tengui l'anotació «@FacesConfig(version = JSF_2_3)» (veure «ConfigurationBean» dins el mòdul «-back» de ProjecteBase).

Les principals novetats són els següents:

- Major suport per HTML5
- Possibilitat d'establir accions per les peticions GET (viewAction)
- Suport per WebSockets
- Implementació de fluxes multipagina (flows)
- Validacions multicamp de formularis
- Suport per la nova API Java Time
- Integració amb CDI (les anotacions dels managed beans passen a estar «deprecated» i s'emprin les anotacions de CDI)
- Major suport AJAX (invocació de mètodes de servidor)
- Suport per pujades de fitxers

En els següents articles trobareu una relació més completa de les funcionalitats afegides a les versions 2.2 i 2.3:

- JSF 2.2: https://arjan-tijms.omnifaces.org/p/jsf-22.html
- JSF 2.3: https://arjan-tijms.omnifaces.org/p/jsf-23.html

4.3.5. CDI

A partir de Java EE 6 es va introduir la nova especificació «Contexts And Dependency Injection», CDI 1.0, que estandarditza el patró d'injecció de dependència dins components Java EE.

Java EE 8 incorpora l'especificació 2.0 de CDI, que s'ha convertit en un element central. La resta d'especificacions s'integren amb aquesta nova especificació de manera que els seus components són «injectables» o poden ser «injectats».

L'especificació completa es pot descarregar aquí: https://download.oracle.com/otndocs/jcp/cdi-2-final-spec/

En aquest enllaç hi trobareu el tutorial oficial de CDI de la documentació de Java EE 8:

- https://javaee.github.io/tutorial/cdi-basic.html

4.3.6. JAX-RS i JSON-B

A partir de Java EE 6 es va incorporar a la plataforma l'especificació JAX-RS per la definició de serveis REST a dins aplicacions web.

Java EE 8 incorpora l'especificació 2.1 de JAX-RS. La llibreria «Jersey» és actualment la implementació de referència de JAX-RS. En el cas de JBoss EAP 7.2, la implementació de JAX-RS es realitza mitjançant la llibreria RestEasy.

Dins Java EE 8 s'ha incorporat també com a novetat la nova API JSON-B que defineix com mapejar objectes a JSON, mitjançant anotacions, una feina per la qual es solia emprar fins ara la llibreria Jackson.

En quasevol cas, la incorporació de JAX-RS implica que podrem definir serveis REST amb JSON emprant anotacions estàndards, sense necessitat d'incorporar cap llibreria específica dins l'aplicació. També incorpora un client estàndard.

L'especificació completa de JAX-RS 2.1 es pot descarregar aquí:

- https://download.oracle.com/otndocs/jcp/jaxrs-2_1-final-eval-spec/index.html,

mentre que l'especificació JSON-B 1.0 es pot descarregar aquí:

- https://download.oracle.com/otndocs/jcp/json_b-1-final-eval-spec/index.html

Dins el mòdul «-api» de ProjecteBase es pot veure un exemple complet de com configurar i crear un servei REST utilitzant JAX-RS i JSON-B.

4.3.7. Bean Validation

A partir de Java EE 6 es va incorporar a la plataforma l'especificació Bean Validation 1.0 per la definició de validacions d'objectes: camps requerits, no buids, validació de rangs de nombres, dates ...

Java EE 8 incorpora la versió 2.0 d'aquesta especificació. La resta d'especificacions incorporen el suport per aquestes validacions. Si anotam els Entity de JPA, aquestes seran validats abans de persistir-los, si anotam els beans emprats dins un formulari amb JSF aquest realitzarà les validacions abans d'enviar el formulari...

Dins ProjecteBase es pot veure l'ús de Bean Validation tant en la capa JPA als objectes que componen el model, com a la capa web amb JSF i els DTO.

4.3.8. Altres especificacions

Altres especificacions noves o millorades dins JBoss EAP 7.2 que poden ser d'interès:

- Actualització a JMS 2.0. L'especificació JMS ha incorporat noves APIs que faciliten molt el seu ús.
- Nova especificació Websocket 1.1. Nova especificació que permet implementar components servidor de websocket.
- Nova especificació Concurrency 1.0. La creació i gestió de threads estava totalment desaconsellada dins els components Java EE, com Servlets i EJBs. La nova especificació «Concurrency Utilities for Java EE» defineix una nova API per la creació i gestió de threads dins Java EE.
- Nova especificació Batch 1.0. Incorpora un conjunt de components per realitzar processos batch.