

**Versione 01****VERIFICHE E APPROVAZIONI**

<b>VERS</b>	<b>REDAZIONE</b>		<b>CONTROLLO APPROVAZIONE</b>		<b>AUTORIZZAZIONE EMISSIONE</b>	
	<b>NOME</b>	<b>DATA</b>	<b>NOME</b>	<b>DATA</b>	<b>NOME</b>	<b>DATA</b>
V01	Antonaci Boschetti Capitani Pavese	12/12/2018	Mauro Antonaci	17/12/2018	Ennio Caggiati	17/12/2018

**STATO DELLE VARIAZIONI**

<b>VERS</b>	<b>PARAGRAFO O PAGINA</b>	<b>DESCRIZIONE DELLA VARIAZIONE</b>
V01	Tutto il documento.	Versione iniziale del documento.

## Sommario

<b>1. SCOPO DEL DOCUMENTO .....</b>	<b>4</b>
<b>2. RIFERIMENTI .....</b>	<b>4</b>
<b>3. CAMPO DI APPLICAZIONE.....</b>	<b>4</b>
<b>4. DESTINATARI ED UTILIZZO DEL DOCUMENTO.....</b>	<b>4</b>
<b>5. REQUISITI/VINCOLI SU CUI SI BASANO LE LINEE GUIDA .....</b>	<b>5</b>
5.1 Stabilità nel tempo del build ed indipendenza da repository pubblico	5
5.2 Possibilità di eseguire il build anche se non in rete CSI	6
5.3 Aderenza agli standard di packaging	6
5.4 Evitare duplicazione dell'artefatto rispetto al repository ivy	6
5.5 Stesse coordinate degli artefatti nel repository locale	6
5.6 Valorizzazione a build-time di parametri dipendenti da ambiente	6
5.7 Richiamabilità del build da Jenkins	7
5.8 Possibilità di eseguire l'analisi del codice tramite sonarqube	7
5.9 Possibilità di eseguire debug remoto	7
<b>6. INTRODUZIONE A MAVEN E IMPOSTAZIONE DELLE LINEE GUIDA.....</b>	<b>7</b>
<b>7. INDICAZIONI PER LA GESTIONE DELLE DIPENDENZE .....</b>	<b>9</b>
7.1 Repository enterprise CSI	9
7.1.1 Puntamento al repository enterprise CSI da ambiente di build locale	10
<b>8. INDICAZIONI PER LA GESTIONE DEI SORGENTI.....</b>	<b>10</b>
<b>9. INDICAZIONI PER IL PACKAGING .....</b>	<b>11</b>
9.1 Packaging della componente	11
9.1.1 Packaging parametrico: indicazioni di configurazione	11
9.1.2 Produzione dell'unità di installazione in formato TAR	13
9.1.2.1 scelta del modulo maven su cui integrare il maven-assembly-plugin.....	13
9.1.2.2 configurazione di un assembler.....	14
9.1.2.3 esecuzione dell'assembler nel ciclo di build.....	15
9.2 gestione del build in locale	15
9.3 Gestione del build tramite Jenkins	16
9.3.1 Definizione di un Job Jenkins per il build di un progetto maven	16
9.3.2 Accorgimenti per pubblicazione di artefatti intermedi	17
9.3.3 Analisi del codice con sonar	18
<b>10. APPENDICE A: VERSIONI DI RIFERIMENTO.....</b>	<b>18</b>
<b>11. APPENDICE B: SETUP AMBIENTE DI SVILUPPO (IN ECLIPSE).....</b>	<b>18</b>
11.1 Installazione di maven sulla postazione di sviluppo	18
11.2 Installazione del plugin M2E in eclipse	18
11.3 Abilitazione delle configurazioni locali di M2E	19
11.4 Configurazione di una cartella differente per il repository locale	19
11.5 Configurazione del proxy	19
<b>12. APPENDICE C: RIFERIMENTI, SEGNALAZIONI E SUGGERIMENTI.....</b>	<b>20</b>

<b>13.</b>	<b>APPENDICE D: PROGETTO DI ESEMPIO .....</b>	<b>20</b>
<b>14.</b>	<b>APPENDICE E: GLOSSARIO .....</b>	<b>21</b>

## 1. Scopo del documento

Questo documento ha lo scopo di illustrare alcune linee guida per l'utilizzo di *Maven* come strumento di build e *dependency management* per lo sviluppo di applicazioni *Java* nella software factory CSI-Piemonte.

Il documento non è un manuale di *Maven*: per la conoscenza approfondita dello strumento si rimanda alla documentazione disponibile on-line,

## 2. Riferimenti

- [1] Maven - getting started:  
<https://maven.apache.org/guides/getting-started/index.html>
- [2] Maven su Wikipedia:  
[https://en.wikipedia.org/wiki/Apache\\_Maven](https://en.wikipedia.org/wiki/Apache_Maven)
- [2] serie di articoli su Mokabyte (datati ma ancora attuali):
  - Filosofia d'uso: <http://www.mokabyte.it/2007/01/maven-1/>
  - Archetipi: <http://www.mokabyte.it/2007/02/maven-2/>
  - Il *pom.xml*: <http://www.mokabyte.it/2007/04/maven-3/>
  - I repository server: <http://www.mokabyte.it/2007/07/maven-6/>

## 3. Campo di applicazione

Queste linee guida sono destinate ad essere utilizzabili nello sviluppo di progetti che seguono gli stack applicativi *java-based* in vigore a partire dal 2017.

Le linee guida si applicano a progetti con le seguenti caratteristiche:

- Nuovi sviluppi con realizzazione completa nella software factory CSI
- Nuovi sviluppi con realizzazione a corpo

I progetti ottenuti come riuso esterno non devono sottostare necessariamente a tutte le presenti linee guida. E' però facoltà del progetto utilizzare una o più indicazioni contenute nelle linee guida, se lo ritengono opportuno/utile. In ogni caso per le indicazioni relative ai progetti in riuso esterno si faccia riferimento alle apposite linee guida.

I progetti che utilizzano Maven, ma sono stati sviluppati precedentemente all'emissione delle presenti linee guida, possono decidere o meno di adottare le indicazioni marcate come "facoltative", mentre saranno tenuti ad adeguarsi alle indicazioni marcate come "obbligatorie" quando possibile.

Le versioni di riferimento sono elencate al par. 9.3.3.

N.B: l'introduzione di *Maven* come strumento di *build* è da considerarsi in affiancamento ad altri strumenti (es. ANT+IVY) e non in sua sostituzione.

## 4. Destinatari ed utilizzo del documento

Questo documento è destinato ai progettisti/sviluppatori che, avendo la necessità di realizzare applicazioni *JEE*, decidano di utilizzare *Maven* come strumento per il *build* e il *dependency management*.

A corredo del documento verranno forniti:

- Un esempio di progetto che implementa un servizio *REST* in tecnologia *JAX-RS* seguendo le presenti linee guida.
- Un template di job utilizzabile per configurare il build jenkins di una componente che utilizza maven

Il modo più efficace di utilizzare queste linee guida pertanto è il seguente:

- Configurare l'ambiente, se necessario seguendo le istruzioni contenute al paragrafo 11
- Scaricare il progetto di esempio
- Leggere i concetti contenuti nel documento facendo riferimento all'esempio

## 5. Requisiti/Vincoli su cui si basano le linee guida

Le presenti linee guida sono ispirate dalla presenza di requisiti o vincoli specifici relativi al contesto di applicazione, ovvero:

- i sistemi informativi degli enti della PA piemontese
- le caratteristiche degli ambienti di esecuzione della server farm CSI
- i processi di sviluppo e rilascio della software factory CSI

In questo capitolo sono elencati tali requisiti/vincoli e nei successivi capitoli si farà riferimento ad essi per giustificare alcune delle scelte fatte. Non è pertanto indispensabile conoscere approfonditamente questo capitolo ai fini dell'applicazione delle linee guida, ma se ne consiglia ugualmente la lettura.

Esistono requisiti/vincoli di varia forza:

- alcuni vincoli sono imprescindibili in quanto il loro mancato rispetto potrebbe rendere non possibile, ad esempio, il dispiegamento dell'applicativo, oppure la corretta gestione del prodotto in esercizio, oppure la gestione del progetto da un punto di vista di processo
- alcuni vincoli sono da considerarsi meno tassativi, in quanto il loro mancato rispetto non comprometterebbe gli aspetti di cui sopra mentre invece il loro rispetto potrebbe garantire, ad esempio, una maggiore efficienza del processo di sviluppo, migliori performance, etc..

Pertanto, per ciascuno dei vincoli verrà anche indicato il livello di *importanza*, oltre che l'aspetto che il rispetto di tale vincolo influenza.

Nei paragrafi seguenti sono elencati i vari requisiti/vincoli e si specifica anche il grado di copertura del requisito fornito da queste linee guida, che potrà essere:

- requisito totalmente soddisfatto
- requisito non soddisfatto o non soddisfacibile (può essere solo per quei requisiti che non siano dichiarati indispensabili)
- requisito non ancora preso in considerazione (sarà preso in considerazione in future versioni di linea guida o in linee guida accessorie)

### 5.1 Stabilità nel tempo del build ed indipendenza da repository pubblico

E' necessario rendere stabile e ripetibile nel tempo il processo di *build*, ed è pertanto necessario rendere il processo indipendente da variazioni non controllate che intervengano sui *repository* pubblici.

- Importanza: obbligatoria
- Elementi influenzati: *build*, gestione dei sorgenti, *dependency management*

Questo requisito risulta totalmente soddisfatto, mediante la predisposizione di un *repository* enterprise basato su *artifactory* pro, secondo quanto descritto al capitolo 7.

## 5.2 Possibilità di eseguire il build anche se non in rete CSI

Deve essere possibile eseguire il *build* sia all'interno della rete CSI, sia quando non si è in rete CSI, con minime necessità di riconfigurazione. Ovvero il passaggio dalla modalità locale fuori CSI a quella locale CSI, a quella centralizzata su build machine, deve richiedere al massimo qualche riconfigurazione (es. url del *repository*).

- Importanza: obbligatoria
- Elementi influenzati: *build*, *dependency management*

Questo requisito risulta soddisfatto solo se si può accedere in VPN alla rete CSI, a causa della necessità di referenziare il *repository maven* enterprise. Solo in tale repository, infatti, saranno disponibili le librerie trasversali prodotte in CSI. Per contro non sarà necessaria alcuna riconfigurazione di puntamenti di maven per far funzionare il build nei vari contesti di lavoro.

## 5.3 Aderenza agli standard di packaging

I pacchetti costruiti come prodotto del processo di *build* devono essere conformi alle specifiche di *packaging* definite in CSI. Ad esempio, deve essere possibile produrre un *TAR* contenente l'*EAR* dell'applicazione e eventuali file di deployment accessori – es. descrittore di *datasource*).

- Importanza: obbligatoria
- Elementi influenzati: *build*

Questo requisito risulta completamente soddisfatto seguendo le indicazioni riportate nel capitolo 9.

## 5.4 Evitare duplicazione dell'artefatto rispetto al repository ivy

Al fine di razionalizzare lo storage del *repository* e le attività di pubblicazione deve essere possibile riutilizzare per la pubblicazione in modalità *Maven* gli artefatti già pubblicati in modalità *Ivy*. Questo vale sia per le pubblicazioni preesistenti che per quelle future.

- Importanza: opzionale
- Elementi influenzati: *build*

Questo requisito risulta soddisfatto in modo indiretto, grazie all'assenza di un processo di pubblicazione nel *repository maven* aziendale, che è invece un *mirror* dei *repository* pubblici su internet.

## 5.5 Stesse coordinate degli artefatti nel repository locale

Il puntamento ad artefatti da parte degli applicativi deve essere tale per cui un artefatto disponibile su un *repository* pubblico sia referenziato con le stesse coordinate anche sul *repository* aziendale.

N.B: questo requisito è subordinato all'adozione di un *repository* locale. Nel caso in cui non si adotti questa soluzione il requisito non ha più ragione di essere posto in quanto garantito per definizione.

- Importanza: obbligatorio
- Elementi influenzati: *build*, *dependency management*

Questo requisito risulta totalmente soddisfatto, poiché il repository aziendale è un *mirror* dei *repository* pubblici.

## 5.6 Valorizzazione a build-time di parametri dipendenti da ambiente

Tipicamente è necessario parametrizzare a tempo di *build* le componenti software, in un modo che varia tra i differenti ambienti (ad. esempio nel caso in cui la componente debba colloquiare con servizi *supplier* occorrerà parametrizzare le coordinate per la connessione a tali servizi dalla logica applicativa).

- Importanza: obbligatoria

- Elementi influenzati: *build*, gestione dei sorgenti

Questo requisito risulta totalmente soddisfatto, grazie all'applicazione del meccanismo dei *profile*, descritto al paragrafo 9.1.1.

## 5.7 Richiamabilità del build da Jenkins

Al fine dell'inserimento nella *tool-chain* di *delivery automation* deve essere possibile attivare il processo di build da *jenkins*.

- Importanza: obbligatorio
- Elementi influenzati: build

Questo requisito risulta essere totalmente soddisfatto, in quanto si tratta di una componente standard *java* / *JEE*, secondo quanto descritto al par. 9.3.

## 5.8 Possibilità di eseguire l'analisi del codice tramite sonarqube

Deve essere possibile eseguire, nell'ottica di *continuous build*, l'analisi dei sorgenti tramite *sonarqube*.

- Importanza: obbligatorio
- Elementi influenzati: build, struttura sorgenti

Questo requisito risulta totalmente soddisfatto secondo quanto indicato al paragrafo 9.3.3.

## 5.9 Possibilità di eseguire debug remoto

La struttura di sorgenti/progetti deve permettere il *debug* remoto dall'IDE Eclipse

- Importanza: obbligatorio
- Elementi influenzati: build e struttura sorgenti

Questo requisito risulta soddisfatto, purchè venga correttamente configurato il path dei sorgenti nella *debug configuration* (di tipo *Remote Java Application*) di *eclipse*.

# 6. Introduzione a Maven e impostazione delle linee guida

*Maven* è uno strumento completo per la gestione di progetti software Java, in termini di compilazione del codice, distribuzione, documentazione e collaborazione del team di sviluppo.

Maven è contemporaneamente:

- un insieme di **standard**,
- una struttura di **repository**
- un'applicazione

che servono alla gestione e la descrizione di progetti software.

Esso definisce un **ciclo di vita standard per il building, il test e il deployment** di file di distribuzione Java.

Gli obiettivi principali di *Maven*, di interesse per le presenti linee guida, sono

1. Semplificazione del processo di *build* dei sistemi *Java*. In particolare, *Maven* si fa carico di risolvere tutta una serie di dettagli senza ricorrere all'utilizzo di file di *script*,
2. Sviluppo di un ambiente uniforme di *build*. *Maven* gestisce progetti basati sul proprio modello a oggetti del progetto (POM, *Project Object Model*). Quantunque molti aspetti di *Maven* possano

essere personalizzati, *Maven* dispone di una serie di "standard", studiati per essere efficacemente utilizzati per i vari progetti. Pertanto, una volta compreso il funzionamento di un progetto gestito da *Maven*, si sa come gestire quasi ogni altro progetto (a meno di forti personalizzazioni).

3. Erogazione di linee guida corredate da un opportuno supporto per l'applicazione di *best practice* per lo sviluppo di sistemi.

Le principali caratteristiche offerte da *Maven*, di interesse per le presenti linee guida, sono:

- *setup* semplificato dei progetti in rispetto alle *best practices* "implementate": preparare un nuovo progetto o estrapolare un modulo è un'operazione molto rapida;
- utilizzo consistente indipendente dai singoli progetti, il che equivale a minimizzare la curva di apprendimento;
- gestione avanzata delle dipendenze corredata da aggiornamento automatico e gestione delle dipendenze transitive ( $A \rightarrow B, B \rightarrow C \Rightarrow A \rightarrow C$ );
- continua espansione della già considerevole disponibilità di *plug-in*, librerie e *meta-dati* da utilizzarsi per la gestione dei propri progetti: molti di questi elementi, inoltre, sono *open-source* e la relativa comunità è molto attiva;
- estensibilità: è possibile non solo modificare le impostazioni di default, ma anche scrivere propri *plug-in*;
- *build* basati su modello: *Maven* è in grado di eseguire il *build* di una serie di progetti e di includere le relative distribuzioni in appositi file .jar, .war etc. senza dover ricorrere a nessuno *script*;

Date le caratteristiche di *Maven* evidenziate nei paragrafi precedenti, i vantaggi derivanti dall'utilizzo nella gestione dei progetti software, che hanno un impatto sulle presenti linee guida, sono principalmente i seguenti:

- Coerenza: le varie organizzazioni possono standardizzare la gestione dei progetti *Java* utilizzando l'insieme di *best practice* alla base di *Maven*. Rispetto a strumenti che non hanno una implementazione intrinseca di linee guida e standard, l'utilizzo di *Maven* permette di definire meno indicazioni specifiche per implementare alcune *best practices*.
- Semplificazione della manutenzione: il tempo necessario per mantenere gli ambienti e gli *script* di *build*, è minore che in altri strumenti (grazie alla standardizzazione implicita e alla implementazione automatica dello strumento).

*Maven* introduce un *pattern* per lo sviluppo dei progetti *software* basato sui seguenti principi base:

1. convenzioni sulla configurazione (*convention over configuration*) e in particolare:
  - a. organizzazione standard della directory dei progetti (ubicazione delle risorse del progetto, dei file di configurazione, di quelli generati, della documentazione, etc.);
  - b. definizione del vincolo base secondo cui un progetto *Maven* genera un solo output (file di distribuzione). Ciò porta naturalmente alla realizzazione di file pom.xml con un obiettivo limitato e ben definito, e quindi all'applicazione del principio di separazione delle responsabilità;
  - c. convenzione sui nomi;
2. esecuzione dichiarativa;
3. riutilizzo di logica di *build*;
4. organizzazione coerente delle dipendenze.

Se si organizza la struttura dei propri progetti in base allo standard *Maven*, quest'ultimo è in grado di fornire automaticamente una serie di servizi, come la generazione dei file di progetto per i vari IDE, la compilazione automatica, etc.

*Maven*, in più, oltre a fornire delle strategie predefinite, permette di modificarle, qualora sia proprio necessario.



Il recepimento di tali indicazioni permette di rendere più snelle le linee guida aziendali che a questo punto non dovranno inventare nulla di nuovo ma fare semplicemente riferimento ad esse.

Per quanto riguarda l'esecuzione dichiarativa (punto 2) basti dire che tutto in *Maven* ha una natura dichiarativa (che tuttavia, volendo, può essere alterata). L'elemento base di *Maven*, il file *pom.xml*, è probabilmente l'esempio più classico di struttura dichiarativa. A differenza di quanto avviene con *Ant*, con poche righe di carattere dichiarativo (e non procedurale), è possibile compilare, verificare, generare la documentazione e il file di distribuzione di semplici progetti.

Per quanto riguarda l'organizzazione coerente delle dipendenze (punto 4), questa è ottenuta attraverso una serie di meccanismi di *repository*. In primo luogo, esiste una sezione all'interno del file *pom.xml*, dedicata alla dichiarazione delle dipendenze.

*Maven* dispone di due tipi di *repository*: *locale* e *remoto*. Durante il normale funzionamento, *Maven*, interagisce con il *repository locale*, dove con "locale" si intende locale alla macchina che esegue il build; qualora però una dipendenza non sia presente all'interno di tale repository, *Maven* si occupa di consultare i repository remoti ai quali ha accesso, al fine di risolvere la dipendenza mancante. Qualora non si definisca diversamente, l'ultimo repository acceduto in ordine di tempo è il repository globale presso ibiblio (<http://mirrors.ibiblio.org/pub/mirrors/maven2/>). Questa consultazione serve per individuare, e quindi scaricare nei vari repository (dapprima quelli remoti e poi automaticamente in quelli locali) i file necessari per risolvere la dipendenza dichiarata (tipicamente degli archivi JAR).

Da questa breve introduzione si evince che il modo più efficace di realizzare linee guida per l'utilizzo di *maven* consiste in:

- affidarsi il più possibile alle implementazioni ed indicazioni implicite di *best practices* messe a disposizione direttamente dallo strumento
- aggiungere esclusivamente le indicazioni necessarie all'implementazione di vincoli o requisiti specifici del contesto CSI-Piemonte

Le presenti linee guida sono stilate ispirandosi a questa impostazione.

## 7. Indicazioni per la gestione delle dipendenze

### 7.1 Repository enterprise CSI

Al fine di assolvere al requisito di ripetibilità del build (v. 5.1) è necessario che la risoluzione delle dipendenze sia effettuata su un repository gestito in CSI (di seguito definito "repository enterprise CSI"), con le seguenti caratteristiche:

- Deve contenere almeno tutte le librerie necessarie per il build dei progetti che aderiscono alle linee guida
- Gli artefatti reperibili sui repository pubblici devono essere referenziabili con le stesse "coordinate" anche nel repository enterprise CSI.
- Il processo di pubblicazione nel repository enterprise CSI deve essere controllato con regole e processi analoghi a quelli predisposti per il repository basato su IVY. A differenza di quanto avviene per IVY, però, con *maven* è prevista la pubblicazione nel repository enterprise CSI delle sole librerie realizzate in CSI, essendo quelle trasversali generalmente reperibili direttamente sui repository pubblici.

Il repository enterprise CSI è configurato come *mirror* che unifica i seguenti *repository*:

- Repository pubblici reperibili su internet (maven-central + eventuali altre fonti)
- Repository locale CSI, per le librerie prodotte in CSI

### 7.1.1 Puntamento al repository enterprise CSI da ambiente di build locale

Per un utilizzo su postazione di sviluppo Windows, per puntare al repository enterprise CSI è sufficiente configurare il file *settings.xml* nella propria cache locale (di default cartella *.m2* nella HOME dell'utente windows) nel modo seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
http://maven.apache.org/xsd/settings-1.1.0.xsd" xmlns="http://maven.apache.org/SETTINGS/1.1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<mirrors>
  <mirror>
    <id>csi-central</id>
    <name>CSI Repart</name>
    <url>http://repart.csi.it/maven2</url>
    <mirrorOf>*</mirrorOf>
  </mirror>
</mirrors>
</settings>
```

In questo modo tutte le operazioni di reperimento degli artefatti (siano essi disponibili su repository pubblici che pubblicati sul repository locale) saranno mediate da tale mirror.

N.B: essendo repart.csi.it visibile solo in rete CSI, per l'utilizzo del repository enterprise CSI è necessario essere collegato alla rete CSI, direttamente o tramite VPN opportunamente configurata, e non è necessario utilizzare il proxy.

## 8. Indicazioni per la gestione dei sorgenti

I progetti che utilizzano maven sono tipicamente organizzati in vari moduli. Tipicamente ciascun modulo può generare un artefatto, che può essere intermedio o finale.

Nell'utilizzo di *maven* per la gestione di progetti che danno origine a componenti software di tipo *JEE*, tipicamente i moduli di progetto che generano un artefatto finale dipendono da uno o più moduli che generano un artefatto intermedio. A loro volta un artefatto intermedio può dipendere da altri artefatti intermedi.

Come anticipato nel cap. 6, *Maven* implementa implicitamente alcune *best-practices* per la strutturazione dei sorgenti, per la produzione degli artefatti, per la gestione della configurazione di *environment*, che discordano necessariamente da quanto previsto nelle linee guida CSI per i progetti che utilizzano ANT + IVY: la maggior parte di tali indicazioni non è bypassabile, in quanto sul tali convenzioni si basa il funzionamento dei plugin che realizzano il *build*. E' pertanto necessario seguire il più possibile le linee guida universalmente note.

Ai fini dell'organizzazione dei sorgenti, per evitare il proliferare di componenti distinti su SVN/GIT, l'indicazione è quella di:

- Inserire in un unico componente SVN/GIT l'intero progetto maven che genera una unità di installazione effettiva (nel caso in cui un prodotto sia costituito da più componenti installabili devono essere creati più progetti maven, ciascuno inserito nella propria *codeline* SVN/GIT).
- All'interno dell'unico componente SVN, strutturare il progetto maven secondo le best practices dello strumento (<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html> )

A titolo esemplificativo si riporta un esempio di struttura di primo livello relativo ad un ipotetico componente *JEE* che implementa lo stack applicativo *CSI Single Page Applications Angular2/Resteasy/spring* nella porzione *server JAX-RS* (basato sulla implementazione *resteasy* di *JBoss*) ed è costituito da:

- un modulo *jar* che definisce le interfacce e l'implementazione del servizio REST
- un modulo *war* che espone il servizio

- un modulo *ear* che assembla il tutto nel componente finale
- Un modulo *tar* che definisce le regole di assemblaggio del pacchetto di installazione in aderenza agli standard di packaging CSI.

```
tstjaxrsmvn/  
  tstjaxrs-jar  
    src  
    ...  
    target  
    ...  
    pom.xml  
  tstjaxrs-war  
    src  
    profiles  
    ...  
    target  
    ...  
    pom.xml  
  tstjaxrs-ear  
    src  
    ...  
    target  
    ...  
    pom.xml  
  tstjaxrs-tar  
    src  
    ...  
    target  
    ...  
    pom.xml  
  
pom.xml
```

come si può notare, rispetto alla struttura standard per ANT+IVY:

- non sono più presenti le cartelle
  - *src* (spostata all'interno dei singoli moduli e organizzate diversamente)
  - *build* (sostituita dalle cartelle “target” all'interno dei singoli moduli)
  - *buildfiles* (le configurazioni specifiche per ambiente sono ora gestite nella cartella *profiles* di ogni modulo e nel relativo *pom.xml*)
  - *dist* (sostituita dalle cartelle target)
- non è più presente il file *build.xml*, sostituito dal file *pom.xml*
- è presente una cartella di primo livello per ogni modulo di cui è costituito il progetto.

## 9. Indicazioni per il packaging

### 9.1 Packaging della componente

*Maven* mette a disposizione vari plugin per eseguire le varie fasi di build di un progetto, che può essere costituito da più moduli. Il tipo di packaging desiderato per il modulo è indicato a livello di *pom.xml*, per ogni tipologia entra in gioco un plugin in grado di produrre l'artefatto di modulo nel packaging scelto (come descritto in [http://maven.apache.org/pom.html#Maven\\_Coordinates](http://maven.apache.org/pom.html#Maven_Coordinates)).

La produzione di un artefatto di modulo è realizzato dal *maven goal* “package”.

#### 9.1.1 Packaging parametrico: indicazioni di configurazione

*Maven* mette a disposizione il meccanismo dei *profiles*, utile per produrre artefatti di modulo la cui configurazione dipende dall'ambiente finale di installazione (come descritto in <https://maven.apache.org/guides/introduction/introduction-to-profiles.html>).

Rimandando per una trattazione dettagliata alla documentazione ufficiale, si descrive di seguito l'utilizzo dei *profiles* nell'applicazione di esempio.

Nel modulo *tstjaxrs-war* dell'applicazione di esempio sono stati introdotti dei profili *maven* per gestire la parametrizzazione di alcune proprietà in funzione dell'ambiente target di rilascio finale; nello specifico, è presente un parametro della risorsa *web.xml* il cui valore dipende dall'ambiente.

Segue una descrizione delle risorse e delle configurazioni aggiuntive per lo scopo.

**Come specificare un profilo:** nel progetto del modulo WAR occorre indicare le proprietà il cui valore dipende dall'ambiente finale:

- Si introduce un file di progetto al path `/profiles/<nome_profilo>/config.properties`, che contiene le properties di configurazione valorizzate per quel *target environment* (nell'esempio sono gestiti due profili per gli ambienti *dev* e *prod-int-01*)

**Dove e come integrare un profilo nel processo di build:** nel nostro esempio si interviene a livello di *pom.xml* di modulo WAR, con le seguenti azioni:

- configurare un *filter* nella sezione *build* che punta alla configurazione del profilo scelto all'atto del lancio del *build maven* (il riferimento è configurato con una proprietà, interpolata a tempo di lancio del build):

```
...
<build>
  <filters>
    <!-- Carico il profilo per l'ambiente target -->
    <filter>profiles/${build.profile.id}/config.properties</filter>
  </filters>
  <plugins>
    ...
  </plugins>
</build>
...
```

- abilitare la sostituzione dei placeholder nelle risorse parametriche rispetto all'ambiente, impostando un flag del *maven-war-plugin*, e la locazione (tipicamente una directory) della risorsa (o collezione di risorse) su cui applicare la sostituzione:

```
...
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
    <webResources>
      <resource>
        <!-- abilita il replace dei placeholder -->
        <filtering>true</filtering>
        <!-- this is relative to the pom.xml directory -->
        <directory>src/main/resources</directory>
      </resource>
    </webResources>
  </configuration>
</plugin>
...
```

- indicare la configurazione dei profili richiesti, tipicamente un profilo per ogni ambiente per cui è necessario effettuare il build del progetto (NOTA: la configurazione può essere applicata con scope

più ampio, ad esempio a livello di *parent pom*, se vogliamo adottare gli stessi profili per ogni modulo, oppure definiti una-tantum nel file *settings.xml* se trasversali a più progetti)

```
...
<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <build.profile.id>dev</build.profile.id>
    </properties>
  </profile>
  <!-- Configurazioni per l'ambiente di produzione -->
  <profile>
    <id>prod-int-01</id>
    <properties>
      <build.profile.id>prod-int-01</build.profile.id>
    </properties>
  </profile>
</profiles>
...
```

### 9.1.2 Produzione dell'unità di installazione in formato TAR

Nel contesto CSI, il requisito di assemblaggio prevede di produrre un archivio finale di installazione in formato tar (o zip) con contenuti che possono variare a seconda della tecnologia di sviluppo della componente

#### 9.1.2.1 scelta del modulo maven su cui integrare il maven-assembly-plugin

E' utile e più ordinato definire un modulo maven dedicato a questo scopo, figlio anch'esso del *parent-pom* di progetto. Esso è così configurato:

```
<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>
  <parent>
    <groupId>it.csi.test</groupId>
    <artifactId>tstjaxrs</artifactId>
    <version>1.0.0</version>
  </parent>
  <artifactId>tstjaxrs-tar</artifactId>
  <packaging>pom</packaging>
  <name>tstjaxrs-tar</name>
  <description>Produce un archivio TAR contenente l'ear di progetto</description>

  <!-- NOTE: These dependency declarations are only required to sort this
    project to the end of the line in the multimodule build. Since we only include
    the tstjaxrs-ear module in our assembly, we only need to ensure this distribution
    project builds AFTER that one... -->

  <dependencies>
    <dependency>
      <groupId>it.csi.test</groupId>
      <artifactId>tstjaxrs-ear</artifactId>
      <version>1.0.0</version>
      <type>ear</type>
    </dependency>
  </dependencies>

  <build>
    <finalName>tstjaxrs-1.0.0</finalName>
    <plugins>
      <plugin>
```

```

<artifactId>maven-assembly-plugin</artifactId>
<version>3.1.0</version>
<executions>
  <execution>
    <id>distro-assembly</id>
    <phase>package</phase>
    <goals>
      <goal>single</goal>
    </goals>
    <configuration>
      <descriptors>
        <descriptor>src/assembly/distribution.xml</descriptor>
      </descriptors>
    </configuration>
  </execution>
</executions>
</plugin>
</plugins>
</build>
</project>

```

## 9.1.2.2 configurazione di un assembler

Le direttive di configurazione possono essere inserite, oltre che nel POM di modulo TAR, anche in un file XML di modulo TAR separato, mantenendolo pertanto distinto; questa è stata la scelta adottata nell'esempio dettagliato di seguito:

- file di configurazione (ad esempio al path *src/assembly/distribution.xml*):

```

<assembly xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.3
http://maven.apache.org/xsd/assembly-1.1.3.xsd">
  <id>1.0.0</id>
  <formats>
    <format>tar</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <moduleSets>
    <moduleSet>
      <includes>
        <include>it.csi.test:tstjaxrs-ear</include>
      </includes>
    </moduleSet>
  </moduleSets>
  <files>
    <file>
      <source>../tstjaxrs-ear/target/tstjaxrs.ear</source>
    </file>
    <file>
      <source>src/tstjaxrs-ds.xml</source>
    </file>
  </files>
</assembly>

```

La precedente configurazione esplicita:

- il formato di output (tar)
- i files .ear di progetto prodotto dalla fase *package* del modulo EAR ed il file di configurazione del *datasource*, che andranno inseriti nel *tar* finale.
- configurazione aggiuntiva del *POM* per referenziare il file che descrive l'assemblaggio custom:

```

...
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>

```

```
<version>3.1.0</version>
<executions>
  <execution>
    <id>distro-assembly</id>
    <phase>package</phase>
    <goals>
      <goal>single</goal>
    </goals>
    <configuration>
      <descriptors>
        <descriptor>src/assembly/distribution.xml</descriptor>
      </descriptors>
    </configuration>
  </execution>
</executions>
</plugin>
...
```

Da notare il **binding** del goal *single* di *plugin maven-assembly* alla fase *package* del ciclo di build complessivo (in quella fase *l'ear* è già stato prodotto dal plugin *maven-ear* e può essere inserito nel *tar* finale).

#### 9.1.2.3 esecuzione dell'assembler nel ciclo di build

Lanciando la fase *package* sul progetto, come ultimo step del ciclo di build parte il goal *single* che produce il *tar* finale. Si riportano alcuni punti di attenzione:

- **nome dell'ear nel tar:** NON deve esserci la versione del componente

motivazione: il deploy dell'ear deve essere una azione singola, cioè si vuole effettuare il solo *reddeploy* sugli Application Server target della filiera produttiva del CSI; se avesse la versione, la persona preposta all'installazione (sia manualmente o tramite automation) avrebbe necessità di compiere due azioni distinte:

- undeploy della versione precedente
- deploy della nuova

Il modo per farlo è impostare l'attributo `<finalName>nome-ear</finalName>` nel POM del modulo *tstjaxrs-ear* (senza estensione).

- **nome del tar prodotto dall'assembly:** il plugin *assembly* aggiunge sempre come suffisso al nome del tar il token `<assemblyId>`, definito nel descrittore di configurazione dell'assemblatore, questa situazione, se tale elemento fosse valorizzato con un identificativo dell'assembly, genererebbe una NON CONFORMITA' rispetto alle regole di naming dei pacchetti di installazione che prevedono che il file finale si chiami `<nome_componente>-<versione>.tar`. Per evitare questo problema è possibile impostare il valore dell'elemento `<assemblyId>` in modo che corrisponda alla versione dell'ear.

## 9.2 gestione del build in locale

Concluse le operazioni di configurazione come indicato nel paragrafo precedente, il packaging della componente in locale avviene richiamando il goal *package* da maven CLI, avendo cura di indicare il profilo richiesto per l'ambiente target:

```
mvn package -P prod-int-01
```



Nel caso non sia fornito esplicitamente un profilo, si attiva il default (nell'esempio il valore di default è: *dev*)

### 9.3 Gestione del build tramite Jenkins

#### 9.3.1 Definizione di un Job Jenkins per il build di un progetto maven

E' disponibile un template specifico per Maven recuperabile, in fase di costruzione di job jenkins, al path:

`/_TEMPLATES/generic_svn_maven325_sonar`

nel quale è necessario aggiornare i parametri che identificano il prodotto – componente oggetto di build.

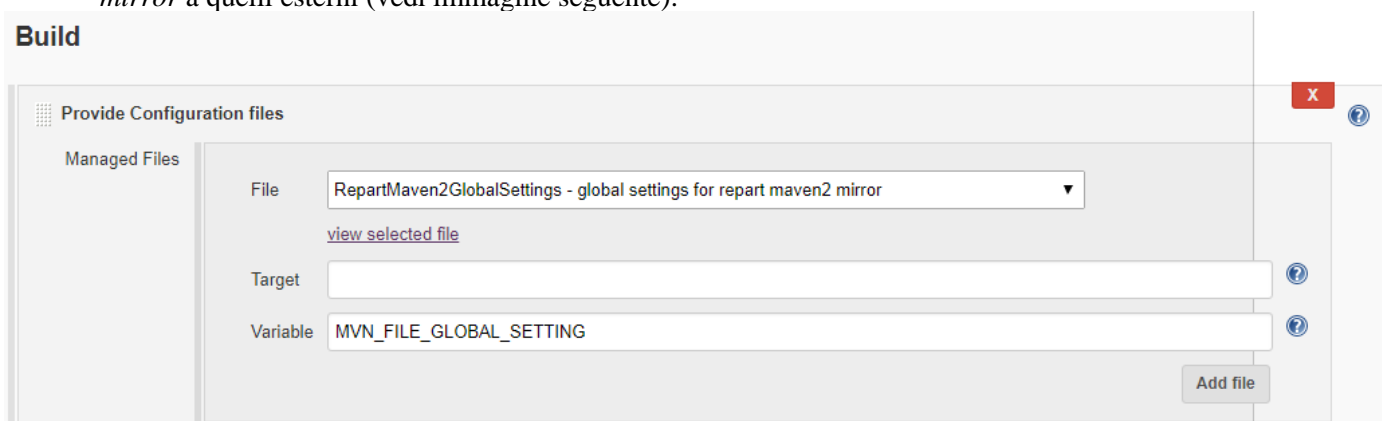
Di seguito si descrivono, a titolo principalmente didattico, i passi per ottenere un job jenkins per effettuare il build di progetti maven a partire da un job di tipo **"Feestyle project"** clonando la configurazione da un template già esistente e adattarlo per inserire l'integrazione con maven e la configurazione sonar alla nuova struttura del progetto di sviluppo.

Il template Jenkins di partenza è:

[http://build.ecosis.csi.it/job/\\_TEMPLATES/job/generic\\_java\\_SVN\\_ant18\\_ivy2\\_jdk18\\_sonar6/](http://build.ecosis.csi.it/job/_TEMPLATES/job/generic_java_SVN_ant18_ivy2_jdk18_sonar6/)

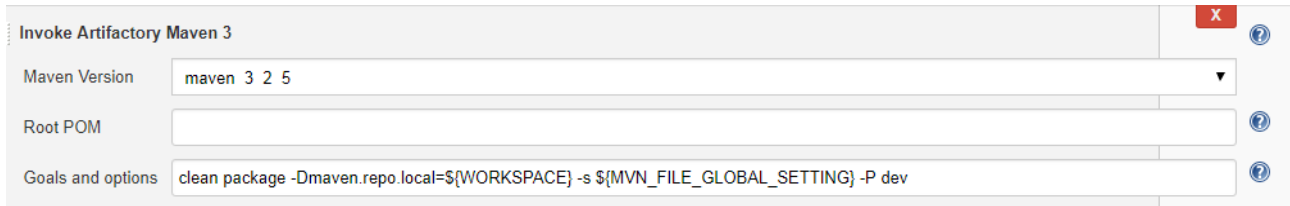
Una volta creato il job, si può procedere alla sua personalizzazione, come dettagliato successivamente:

- sezione build environment => rimosso il flag "Copy files into the job's workspace before building"
- per abilitare l'uso del repository maven locale al job (escludendo quello condiviso a livello di piattaforma di build Jenkins), seguire le indicazioni della Linea Guida per job di tipo *free style project*
- sezione build => inserire lo step "provided configuration files" per impostare un file di settings custom introdotto in piattaforma Jenkins che configura un virtual repository locale con funzione di server *mirror* a quelli esterni (vedi immagine seguente):



- sezione build => rimosso lo step "invoke Ant" *target=dev -lib apache-ivy-2.0.0*
- sezione build => inserito lo step "Invoke Artifactory Maven 3" (vedi immagine seguente), impostando
  - la versione *maven 3.2.5*
  - I goals *clean package*
  - La proprietà *maven.repo.local* a puntare al workspace della componente
  - L'opzione *-s* con la variabile che contiene il path al file setting custom
  - L'opzione *-P* per indicare Il profilo di build da usare
  - lasciare vuoto il campo *ROOT pom* (sarà analizzato il *pom.xml* a livello root della componente)





- sezione build => rimosso lo step finale "invoke Ant" *clean-all -lib apache-ivy-2.0.0*
- adeguamenti della configurazione di integrazione Sonar:
  - commentate alcune delle linee dello script shell di configurazione del client sonar lato jenkins

```
# LIB_LIST=`ls -l lib/ | sed -e "s/^/$esc_WRKSP\/lib\/" | sed -e ':a;N;$!ba;s/\n/,/g'`  
# TAR_FILE=`ls -l dist/${ANT_TARGET}`
```

- modificato il path dei src (riflette la nuova alberatura dei sorgenti):

```
echo "sonar.sources=tstjaxrs-jar/src/main/java" >> sonar-project.properties  
echo "sonar.java.binaries=tstjaxrs-jar/target/classes,tstjaxrs-web/target/classes" >>  
sonar-project.properties  
echo "sonar.java.libraries=tstjaxrs-ear/target/tstjaxrs/lib/*.jar" >> sonar-  
project.properties
```

### 9.3.2 Accorgimenti per pubblicazione di artefatti intermedi

A partire dal 29/05/2017 è stata inibita la scrittura sul repository **.m2 globale** condiviso dagli slave Jenkins.

Si è deciso di implementare questa modifica per adeguarsi ad una delle best practice della continuous integration che prevede di eseguire i build in isolamento usando repository **.m2 locali** al posto di quello globale (v. <http://blog.sonatype.com/2009/01/maven-continuous-integration-best-practices/>).

Di conseguenza tutti i JOB che eseguono build basati su maven (ad esempio quelli necessari per creare i package *.car* per **ws02**) che prevedono il goal **"install"**, dovranno essere opportunamente adeguati, in quanto richiedono l'installazione nel repository di artefatti di build intermedi.

Le variazioni da apportare alla configurazione del job sono le seguenti:

- Per i job di tipo **"Maven Project"** nativo: sezione **Build -> Advanced**, selezionare **Use private Maven repository** ed impostare come **Strategy = Local to workspace**
- Per i job di tipo **"Free Style Project"**: sezione **Build -> Invoke Maven 3**, campo **Goals and Options** aggiungere l'opzione `-Dmaven.repo.local=${WORKSPACE}`

### Nota Bene

per i build che costruiscono il CAR WSO2 bisogna anche aggiungere ai **pom.xml** dei progetti **esb** e **registry** nel blocco "arguments" del plugin exec-maven-plugin l'argomento:

```
<argument>-Dmaven.repo.local=${maven.repo.local}</argument>
```

### 9.3.3 Analisi del codice con sonar

Nel job di build esiste una sezione “execute shell” in cui è definita la configurazione delle proprietà che pilotano il funzionamento del plugin Sonar invocato nello step di build successivo del job: modificato il path dei sorgenti, classi compilate e librerie (riflette la nuova alberatura dei sorgenti) per l’analisi da parte di Sonar 6

## 10. Appendice A: versioni di riferimento

Libreria / strumento	Versione	note
Maven	3.2.5	E’ la stessa versione attualmente presente nella toolchain di continuous building
M2E	1.8.0	Necessario per l’integrazione in eclipse
M2E-WTP connector	1.3.2	Necessario per gestire i progetti che generano jar/war/ear

## 11. Appendice B: setup ambiente di sviluppo (in eclipse)

Per utilizzare maven per lo sviluppo in locale effettuato tramite Eclipse<sup>1</sup> è necessario effettuare il setup dell’ambiente.

1. installazione di maven sulla postazione di sviluppo
2. installazione del plugin M2E in eclipse: da “eclipse marketplace” cercare “M2E” e selezionare tra i risultati “Maven integration for eclipse”
3. (opzionale) configurazione di una cartella differente per la cache locale
4. configurazione del puntamento al repository enterprise CSI, secondo le indicazioni descritte al paragrafo 7.1.1.

### 11.1 Installazione di maven sulla postazione di sviluppo

Al fine di utilizzare durante le attività di sviluppo la stessa versione di maven prevista dalla tool-chain di build, è consigliato installare sulla postazione di sviluppo la stessa versione di maven (v. cap. 9.3.3).

- Scaricare la distribuzione di maven: <https://archive.apache.org/dist/maven/maven-3/3.2.5/binaries/>
- Scompattare in una cartella a piacere

Il tool può essere utilizzato da linea di comando oppure integrato in eclipse, come spiegato nel paragrafo successivo.

### 11.2 Installazione del plugin M2E in eclipse

Non vi sono particolari indicazioni per l’installazione di questo *plugin*. E’ sufficiente ricercarlo nell’*eclipse-marketplace* ed installarlo secondo le consuete modalità.

Per fare in modo che il *plugin* utilizzi l’installazione di *maven* effettuata al punto precedente è necessario configurarla nel dialog di preferenze a cui si accede tramite “window” → “preferences” → “maven” → “installations” (aggiungere una nuova installazione puntando alla cartella in cui è stata scompattata la distribuzione di *Maven* e impostare quella installazione come installazione attiva).

<sup>1</sup> Al momento della scrittura del documento la versione di eclipse di riferimento è Eclipse-Neon

E' inoltre necessario installare, a seconda dei casi, alcuni componenti aggiuntivi, come ad esempio M2E-WTP Maven integration for WTP (v. 1.3.8): permette di gestire artefatti JEE.

### 11.3 Abilitazione delle configurazioni locali di M2E

Nell'utilizzo di maven in *eclipse*, tramite il plugin M2E, è possibile abilitare configurazioni personalizzate globali o a livello di utente windows. Per far ciò è necessario:

- Specificare la posizione del file "settings.xml", all'interno della finestra accessibile tramite "window" → "preferences" → "maven" → "user settings", nel campo "global settings" o "user settings".
- Creare il file del punto precedente, contenente ad esempio quanto riportato nell'esempio seguente:

```
<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">

  <interactiveMode/>
  <usePluginRegistry/>
  <offline/>
  <pluginGroups/>
  <servers/>
  <mirrors/>
  <proxies/>
  <profiles/>
  <activeProfiles/>
</settings>
```

E' importante ricordarsi che per rendere effettive le variazioni è necessario effettuare l'update dei progetti, tramite l'azione "maven" → "update projects" disponibile cliccando con il tasto destro del mouse sui progetti interessati.

### 11.4 Configurazione di una cartella differente per il repository locale

In alcuni casi potrebbe essere utile modificare l'impostazione di default della cartella del repository locale, nella quale verranno scaricati gli artefatti durante il processo di build.

Di default tale repository è mantenuto nella cartella <user-home>/.m2/repository; un motivo per variare la collocazione del repository rispetto alla posizione di default potrebbe essere rappresentato dalla necessità di utilizzare una partizione disco più capiente.

Per impostare una collocazione personalizzata è necessario modificare come segue il file settings.xml, aggiungendo l'elemento "localRepository":

```
<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">

  <localRepository>D:/maven/repo</localRepository>
  <interactiveMode/>
  <usePluginRegistry/>
  <offline/>
  <pluginGroups/>
  <servers/>
  <mirrors/>
  <proxies/>
  </proxies>
  <profiles/>
  <activeProfiles/>
</settings>
```

### 11.5 Configurazione del proxy

Non è necessario in quanto le indicazioni prescrivono l'utilizzo del repository aziendale CSI, accessibile da rete interna o VPN senza utilizzo del proxy.

## 12. Appendice C: Riferimenti, segnalazioni e suggerimenti

Queste linee guida e i progetti di esempio allegati sono disponibili su [gitlab.csi.it](https://gitlab.csi.it).

La responsabilità della redazione ed evoluzione di queste linee guida è del gruppo standard di produzione (2018).

Le linee guida sono realizzate utilizzando un metodo di redazione collaborativa ed “open” che prevede la disponibilità pubblica dei semilavorati fin dalle prime fasi della lavorazione, al fine di permettere una migliore condivisione e un maggior coinvolgimento dei destinatari.

Le linee guida sono disponibili in lavorazione sul repository gitlab:

[https://gitlab.csi.it/technical-components/tools/maven/LG\\_Maven](https://gitlab.csi.it/technical-components/tools/maven/LG_Maven)

Il progetto di esempio è disponibile in lavorazione sul repository gitlab:

<https://gitlab.csi.it/technical-components/tools/maven/tstjaxrsmvn>

E' possibile segnalare richieste di evoluzione, errori, imprecisioni utilizzando il sistema di issue tracking di gitlab, che per i progetti in oggetto è disponibile all'url:

<https://gitlab.csi.it/groups/technical-components/tools/maven/-/issues>

dove è anche possibile vedere l'elenco delle segnalazioni aperte e lo stato di avanzamento.

Lo stato consolidato delle linee guida è fissato mediante il meccanismo dei tag. Pertanto, allo stato attuale, la baseline di riferimento di queste linee guida è la seguente:

- LG\_Maven: 1.0.0
- tstjaxrsmvn: 1.0.0

## 13. Appendice D: Progetto di esempio

Il progetto di esempio da cui sono tratti gli snippet di codice contenuti in queste linee guida è disponibile sul repository gitlab:

<https://gitlab.csi.it/technical-components/tools/maven/tstjaxrsmvn>

Implementa un semplice servizio jaxrs/reteasy, installabile su application server Jboss 6.4.5.

Per gli scopi delle presenti linee guida è sufficiente verificare la corretta costruzione del pacchetto, che si realizza richiamando il goal *package* con il profilo *dev*, che costruisce il tar di distribuzione.

Nel caso si volesse però verificare anche il deploy ed il funzionamento, le API implementate dal progetto sono le seguenti:

- /ping (GET)
- /soggetti?cognome=...&nome=... (GET)
- /soggetto?cf=<cod.fisc> (GET)
- /soggetto (PUT)
- /soggetto/<cod.fisc> (DELETE)

### 14. Appendice E: Glossario

- Repository locale: nella terminologia di *maven* il repository locale è un repository maven presente sulla postazione di sviluppo (o più in generale sulla macchina dove viene eseguito il comando mvn). Di fatto rappresenta una sorta di cache di uno o più repository remoti.
- Repository remoto: nella terminologia *maven* il repository remoto è un repository disponibile in rete che può essere considerato il *master* degli artefatti. E' da considerarsi "remoto" se non è sulla postazione di sviluppo o più in generale sulla macchina che esegue il comando mvn).
- Repository enterprise CSI: repository maven messo a disposizione dalle piattaforme a supporto dell'ALM in CSI, implementato tramite *artifactory pro* e che realizza un *mirror* di alcuni dei principali repository pubblici (es. maven-central) e del repository locale CSI. Relativamente alla terminologia *maven* questo repository si può considerare come *repository remoto*, (in quanto non è sulla postazione utente) e *corrisponde* al *repository artifactory* identificato con: *Maven2*.
- Repository aziendale CSI: alias di Repository enterprise CSI.
- Repository locale CSI: repository maven sul quale possono essere pubblicati gli artefatti realizzati in CSI (es. librerie client, librerie trasversali realizzate in CSI, ...). Relativamente alla terminologia *maven* questo repository si può considerare come *repository remoto*, (in quanto non è sulla postazione utente) e *corrisponde* al *repository artifactory* identificato con: *mvn-local-repo*.