Creazione progetto da template, customizzazione e produzione war (versioni non standard)

Questa guida ricalca in parte questa, la quale è dedicata alle versioni rilasciate del framework. Questa si differenzia in quanto permette di generare un war a partire da versioni del framework *non standard* o comunque in un qualche modo *speciali*. Queste versioni NON sono rilasciate ufficialmente. Ciò comporta che non esse non sono ne taggate nel *repository git*, ne dispiegate su *Artifactory*. Un esempio della necessità di queste versioni speciali può essere il caso in cui, il rilascio sul cliente, magari contingentato in tempi dettati da fattori esterni, prevede la presenza di nuove funzionalità, che si prevede verranno rilasciate solo in una futura release stabile del framework. Di fatto i tempi sono ignoti dato che si dovrà schedulare la fase di test, etc.. Non utilizzando materiale dispiegato su Artifactory, il build del war avviene tramite direttamente tramite codice sorgente disponibile in locale sulla macchina. Questo codice sorgente, che fa riferimento ad uno specifico branch del framework (e quindi, indirettamente ad una version), va quindi preventivamente scaricato in locale. Verrà spiegato sotto come. Il nome del branch del sorgente sarà tipicamente qualcosa del genere *support/4.5-SPACE*, dove *support/* è fisso, *4.5* si riferisce alla version dove si presume che le feature contenute verranno rilasciate e considerate stabili, *-SPACE* è variabile.

Operazioni preliminari

Prima di procedere, ci si deve assicurare di aver installato nel proprio pc sia Apache Maven (Installazione di Maven) sia Git (Installazione di Git).

La guida sottostante può essere utilizzata in tutto od in parte a seconda si parta da zero o si voglia per esempio rigenerare il war in seguito al rilascio di nuove funzionalità sul branch support di commessa.

Gli esempi sottostanti partiranno dal presupposto che esista un branch di commessa chiamato con il pattern: **[customer_code]+'_'+[module_name]** Per esempio: *cst_space*.

Prima di partire occorre conoscere il nome esatto che è presente nel branch di commessa in tutti i *pom.xml* interessati, dentro il tag <version>:

```
<version>4.5-SPACE</version>
```

a noi interessa

```
4.5-SPACE
```

Passaggio 1: Creazione struttura cartelle di base

Deve esistere la seguente struttura di cartelle:

```
    cst_space
        geowebframework                //folder creata in automatico//
        project
            internal_war
```

```
                warname
                    [vari file..]
                    pom.xml        //file//
                buildpom&makegwwar.bat    //file//
                makegproj.bat        //file//
                pom.xml            //file//
            svil_war
            test_war
            prod_war
            .git                //file//
            .gitignore            //file//
            README.md            //file//
```

La folder **cst_space**, può avere un nome qualsiasi, ma per convenzione verrà chiamata come il repository di commessa. La folder **geowebframework**, **NON VA CREATA MANUALMENTE**, ma verrà generata in automatico dal *buildpom&makegwwar.bat*, una volta opportunamente configurato. La folder **project** sarà il vero e proprio repository di commessa, il nome è fisso (in quanto usato a fuoco nel *buildpom&makegwwar.bat*). Si è scelto di tenere la folder del repository di commessa separata da geowebframework in via prudenziale. La presenza di un repository git dentro un altro repository git potrebbe condurre a problematiche in ambienti *Linux*, anche in caso di *.gitignore* configurato. I file *.git*, *.gitignore*, *README.md* sono tipici e denotano che questa è la folder del repository di commessa. Sotto *project* possono esserci un numero variabile di folder, in base alla necessità. Vedi guida sul Repository dio Commessa. In genere si prevede una folder per l'ambiente di test interno alle infrastrutture GeowebItalia, nel nostro esempio **internal_war**, ed una o più folder per la produzione dei war nei vari ambienti, il cui numero è legato al cliente ed agli accordi preso con esso. La folder **warname** verrà generata utilizzando il file *makegproj.bat* Il file **warname/pom.xml** dovrà essere customizzato secondo le linee guida sottostanti Il file *makegproj.bat* verrà utilizzato per produrre la folder **warname** Il file *pom.xml*, dovrà essere scaricato e configurato secondo le linee guida sottostanti Il file *buildpom&makegwwar.bat* produrrà il war vero e proprio.

Proseguiamo ora supponendo di voler generare il file .war dell'ambiente interno: **internal_war**.

Passaggio 2: Generazione della struttura di base del war (esecuzione del file makeproj.bat)

La prima cosa da fare per creare un progetto maven di base nel proprio pc, è cliccare sul seguente link e avviare il download della cartella ZIP

makegproj.zip

. Dopo aver decompresso makegproj.zip salvare il file makeproj.bat nel cartella che dovrà ospitare il nuovo progetto (nel nostro esempio *internal_war*) Eseguirlo con un doppio click. Verrà aperta una finestra di *shell*, che richiederà due parametri:

- il nome del nuovo progetto maven da generare, nel nostro esempio **warname**,
- la versione di GeowebFramework da scrivere nel pom.xml interno al progetto maven. Qui scegliamone una qualsiasi di quelle certamente disponibili su Artifactory. Esempio: *4.4.7*. Serve solo per produrre il progetto maven. Verrà in seguito cambiata a mano e riallineata con quella del pom.xml sotto la folder *geowebframework*.

Alla fine del processo, la finestra si chiuderà automaticamente, e si potrà vedere la cartella con il nome del progetto maven scelto (cioè quello inserito precedentemente in makeproj.bat). Quest'ultima conterrà tutti i file di configurazione e le librerie necessarie.

Il file BAT, quindi, è utilizzato *una tantum*, cioè solo nella fase iniziale della creazione del nuovo progetto.

Passaggio 3: Aggiunta e configurazione del 'pom.xml build sorgente'

Sotto la folder *internal_war*, va creato il file **pom.xml**. Questo di differenzia dagli altri file con stesso nome che possiamo trovare nelle varie cartelle (*geowebframework/pom.xml* e *warname/pom.xml*). Questo pom serve per buildare il codice sorgente presente sotto la folder *geowebframework* e per produrre il war *warname*, effetto secondario ottenuto aggiungenedo *warname* come modulo.

Possiamo scaricare un template di pom.xml qui.

Una volta copiato il file nella giusta posizione passiamo ad editarlo. Dobbiamo assicurarci che il contenuto del tag <version> sia corrispondente alla version del branch. Nel nostro caso dovremmo già avere 4.5-SPACE

```
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.geowebframework</groupId>
    <artifactId>com.geowebframework</artifactId>
    <version>4.5-SPACE</version>
    <packaging>pom</packaging>
    <name>GeoWebFramework</name>
    <description>GeoWebFramework</description>
```

Inoltre bisogna scegliere quali moduli del sorgente vogliamo buildare. Commentando tramite <!– –> eviteremo la build.

IMPORTANTE Questa configurazione deve essere eseguita in concordanza della configurazione del *warname/pom.xml*. Infatti, se un certo plugin serve al war, esso deve essere dichiarato sia dentro *warname/pom.xml* e presente (non commentato) su questo *pom.xml*.

```
    <modules>
        <module>../../geowebframework/webclient</module>
        <module>../../geowebframework/dataservice</module>
        <module>../../geowebframework/metadataservice</module>
        <module>../../geowebframework/transfer-objects</module>
        <!--<module>../../geowebframework/workflowservice</module>-->
        <module>../../geowebframework/calendar</module>
        <module>../../geowebframework/classificationplugin</module>
        <!--<module>../../geowebframework/ThreeDVisualizer</module>-->
        <module>../../geowebframework/umplugin</module>
        <!--<module>../../geowebframework/cde</module>-->
        <!--<module>../../geowebframework/gwMnemonicCode</module>-->
        <!--<module>../../geowebframework/googleStreetView</module>-->
        <!--<module>../../geowebframework/thematism</module>-->
        <module>../../geowebframework/fornitureplugin</module>
```

```
        <module>warname</module>

    </modules>
```

La parte **../../geowebframework/** va considerata fissa, e dovuta alla struttura delle cartelle. Il resto sono nomi di maven module effettivamente presenti nel codice sorgente dentro *geowebframework*.

Da notare che alla fine è presente <module>warname</module>. Questo farà si che al momento giusto verrà buildata anche la nostra cartella di progetto maven precedentemente creata, e verrà prodotto un il war che ci interessa

Passaggio 4: Personalizzazione pom.xml dentro il progetto maven (warname/pom.xml)

La folder **warname** contiene un file **pom.xml**. E'necessario modificare, con un editor di testo, il contenuto del tag <com.geowebframework.version>, con lo stesso nome del tag <version> del branch di commessa: *4.5-SPACE* nel nostro esempio da

```
    <com.geowebframework.version>4.4.7</com.geowebframework.version>
```

a

```
    <com.geowebframework.version>4.5-SPACE</com.geowebframework.version>
```

Nel **pom.xml** vanno inserite anche le dipendenze a eventuali altri plugin di Geoweb necessari all'esecuzione del proprio progetto, come ad esempio:

```
<dependency>
    <groupId>com.geowebframework</groupId>
    <artifactId>ispplugin</artifactId>
    <version>1.0.0</version>
</dependency>
```

In aggiunta, devono essere eventualmente modificati anche alcuni file di configurazione. In particolare, nel percorso *[nome_progetto]\src\main\resources* ci sono:

- Il file **configuration.properties**, che raccoglie tutti i dati configurabili del progetto, che rappresentano entità costanti, come ad esempio i dati di accesso al database; anche in questo caso, ogni parametro è memorizzato come una coppia di stringhe, una contiene il nome del parametro (cioè la chiave) e l'altra memorizza il valore.
- Il file **log4j.properties**, che contiene tutte le informazioni (codificate come coppie chiave-valore) necessarie alla scrittura dei log di Maven. Il log, infatti, è uno strumento molto utile che va configurato nel modo corretto: esso raccoglie gli output dei diversi processi e, analizzando questi output, permette di capire cosa è accaduto in caso di errori.
- Il file **remapConfiguration.properties** che ha lo scopo di mappare i nomi delle proprietà all'interno del **configuration.properties** con nomi di variabili d'ambiente o di sistema.
- Il file **webconfig.ini** che serve per alcune configurazioni della piattaforma Mapguide.

Invece, nel percorso *[nome_progetto]\src\main\webapp\WEB-INF* si trovano:

- Il **dispatcher-servlet.xml**, che serve da controller per le applicazioni web basate su Spring. Il contenuto è standard, ma, in particolari ambienti, alcuni tag non vengono usati e devono essere cambiati.
- Il file **web.xml**, che descrive l'esecuzione dell'applicazione web e contiene configurazioni per l'accesso ai database.
- Infine, il file **spring-security.xml**, che raccoglie informazioni di autenticazione e di connessione al metadata-source.

Le modifiche apportate ai vari file dipendono dal progetto, ma anche dall'ambiente in cui il progetto deve essere utilizzato.

Passaggio 5: produzione del WAR

Terminate tutte le operazioni di configurazione e personalizzazione del progetto, nella cartella da cui si vuole produrre il WAR, bisogna assicurarsi di avere sotto la cartella dell'ambiente specifico (*internal_war*), a seconda dell'ambiente, uno dei seguenti file:

- buildpom&makegwware.bat (Window)
- buildpom&makegwware.sh (Linux)

E' possibile fare il download della seguente cartella compressa, che li contiene entrambi:

buildpom_makegwware.zip

.

Doppio click per eseguire il file. Si aprirà una finestra di *shell* che mostrerà l'avanzamento dell'operazione. Al termine della procedura, nella finestra verrà riportato un messaggio di successo se tutto è andato a buon fine, oppure un messaggio di errore in caso di eventuali problemi. Il file WAR viene automaticamente salvato all'interno si warname/target.