Creazione di un progetto da template, customizzazione e produzione dei WAR

Utilizzando questa procedura, è possibile creare autonomamente i file WAR di cui si ha necessità partendo da un template già esistente su Artifactory.

E' vivamente consigliato, anche se non mandatorio, che tale procedura venga attuata nelle folder appositamente predisposte nella versione locale del *Repository di Commessa*, descritto qui.

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

Passaggio 1: Generazione della folder del progetto maven (esecuzione del file makeproj*.bat)

Procedura

La prima cosa da fare per creare un progetto di base nel proprio pc cliccare sul seguente link e avviare il download della cartella ZIP, in base alla versione major del progetto da generare:

- makegproj-4.6.x-release.zip
- makegproj-4.6.x.zip (PRE-RELEASE, @Deprecated)
- makegproj-4.5.x.zip
- makegproj.zip

(per le versioni precedenti alla 4.5.X)

Dopo aver decompresso makegproj-4.6.x.zip, salvare il file makegproj-4.6.x.bat nella cartella del repository di prodotto/commessa. Step:

- doppio click su makegproj-X.bat
- inserire il nome del nuovo progetto maven da generare. Es: **mvn_project_name**
- inserire la versione di GeowebFramework, scelta tra quelle disponibili su Artifactory. Es:
  - **4.6.0**
  - **4.6.0-SNAPSHOT**
  - **4.5.4**
  - **4.4.19**
  - **4.4.19-HOTFIX**

La finestra di *shell* mostrerà l'avanzamento del download.

Alla fine del processo, la finestra si chiuderà automaticamente e si avrà una cartella con il nome del progetto (cioè quello inserito precedentemente in makeproj.bat) che conterrà tutti i file di configurazione con setting di base.

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

Passaggio 2: Personalizzazione dei file di configurazione

La prima operazione da fare è intervenire sul file **pom.xml**:

- verificare la **<com.geowebframework.version>** di GeowebFramework (che deve coincidere con quella usata nel makegproj.bat)
- dichiarare le **<dependency>** da utilizzare, comprensive di versionamento:

```
<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 **log4j2.properties** (**log4j.properties**, pre 4.6.x), 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. Utile per configurare quale spring-security.xml utilizzare
- 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.

Note war versioni 4.6.x-release

Sono provvisti nel modello 3 file spring-security.xml

1. **spring-security.xml** (senza keycloak, da arricchire con eventuale configurazione Identity Provider del cliente)
2. **spring-security-keycloak.xml** (specifico per keycloak)
3. **spring-security-deprecated.xml** (senza keycloak, con vari meccanismi di autenticazione usati in passato, anche @Deprecated)

Di default è utilizzato lo *spring-security-keycloak.xml* Questa configurazione può essere modificata nel file **web.xml**

```
<!-- Spring Security STARTS -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        <!-- TODO CONFIGURE, choose one between: -->
        <!-- /WEB-INF/spring-security.xml -->
        <!-- /WEB-INF/spring-security-keycloak.xml -->
        /WEB-INF/spring-security-keycloak.xml
    </param-value>
</context-param>
```

Lasciare solo lo spring-security*.xml di interesse, eliminando gli altri, e verificare la coerenza nel web.xml

Nel **pom.xml** sono commentati tutti i plugin disponibili allo stato attuale nel repository del framework. Come sempre, decommentare i necessari moduli ed eliminare gli altri. Aggiungere inoltre le dipendenze di eventuali repository di prodotto o di commessa.

Note war versioni 4.5.x

Le versione 4.5.0-SNAPSHOT era, secondo corrette politiche aziendali, destinata solo allo sviluppo. Logiche di commessa hanno portato di fatto all'utilizzo di questa versione. Il dispiegamento sui clienti ha forzato anche il relativo versionamento: 4.5.1, 4.5.2, etc.. Per le versioni 4.5.X **non è stato prodotto uno specifico makegproj.bat**. In attesa di uno specifico, la prassi da seguire è utilizzare il **makegproj.bat** precedente, avendo cura di modificare lo spring-security.xml in alcune parti

dentro il tag

```
<http auto-config="true" use-expressions="true" disable-url-
rewriting="true">
...
</http>
```

va aggiunta la parte:

```
<intercept-url pattern="/resources/#{
```

```
T(com.geowebframework.metadataservice.registry.GwReleaseInfos).getRevision()
}/fontawesome/**" access="permitAll" /> <!--issue #509-->
<intercept-url pattern="/resources/#{
T(com.geowebframework.metadataservice.registry.GwReleaseInfos).getRevision()
}/images/favicon.png" access="permitAll" />
<intercept-url pattern="/resources/#{
T(com.geowebframework.metadataservice.registry.GwReleaseInfos).getRevision()
}/images/${images.login.form.logo:gw4_enterprise.png}" access="permitAll" />
<intercept-url pattern="/resources/#{
T(com.geowebframework.metadataservice.registry.GwReleaseInfos).getRevision()
}/${images.IndexBackground:IndexBackground.jpg}" access="permitAll" />
```

immediatamente sotto la parte:

```
<intercept-url pattern="/resources/#{
T(com.geowebframework.metadataservice.registry.GwReleaseInfos).getRevision()
}/icons/**" access="permitAll" />
```

Senza il fix potrebbe comparire navigando la schermata:

```
/*!
 * Font Awesome Free 6.2.0 by @fontawesome - https://fontawesome.com
 * License - https://fontawesome.com/license/free (Icons: CC BY 4.0, Fonts: SIL OFL 1.1, Code: MIT License)
 * Copyright 2022 Fonticons, Inc.
 */
.fa {
  font-family: var(--fa-style-family, "Font Awesome 6 Free");
  font-weight: var(--fa-style, 900); }

.fa,
.fa-classic,
.fa-sharp,
.fas,
.fa-solid,
.far,
.fa-regular,
.fab,
.fa-brands {
  -moz-osx-font-smoothing: grayscale;
  -webkit-font-smoothing: antialiased;
  display: var(--fa-display, inline-block);
  font-style: normal;
  font-variant: normal;
  line-height: 1;
  text-rendering: auto; }

.fas,
.fa-classic,
.fa-solid,
.far,
.fa-regular {
  font-family: 'Font Awesome 6 Free'; }

.fab,
.fa-brands {
  font-family: 'Font Awesome 6 Brands'; }

.fa-1x {
  font-size: 1em; }

.fa-2x {
  font-size: 2em; }

.fa-3x {
  font-size: 3em; }
```

Passaggio 3: produzione del WAR

Terminate tutte le operazioni di configurazione e personalizzazione del progetto maven, bisogna assicurarsi di avere sotto la cartella dell'ambiente specifico, a seconda dell'OS, uno dei seguenti file:

- makegwar.bat (Window)
- makegwar.sh (Linux)

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

makegwar.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 **mvn_project_name/target**.

Creazione Webadmin war

(Dalle 4.6.12) L'eccessivo ingombro degli artefatti webadmin (circa 250MB), che andavano a finire su Artifactory ad ogni tag, combinato al contestuale cambiamento delle policy di rilascio che ha previsto un tag esclusivo per ogni fix (sulla quarta cifra di versionamento 4.6.x.y), ha spinto per la trasformazione del modulo maven gw-webadmin in jar, in maniera che potesse essere inglobato in un pom.xml generico per la creazione del war.

Analogamente alla metodologia di produzione del war del webclient, è disponibile un template da cui partire

*

makewebadmin-4.6.x.zip

In generale non ci sarà mai bisogno di aggiungere maven dependency al pom.xml.

Le dipendenze che erano supportate precedentemente continuano ad esserlo Widget e servizi interni al progetto geowebframework di:

1. gw-advanced
2. gw-cms
3. gw-approvals
4. gw-mnemonic-code
5. gw-3d-visualizer
6. gw-workflow

Widget e servizi esterni al progetto geowebframework

1. gw-tree-collection (2.1.3)

Widget e servizi deprecati:

1. postevita
2. gw-smartplatform

Se necessario ulteriori plugin di commessa/progetto che coinvolgano widget/tipi di menu possono essere sviluppati con la medesima modalità. L'unica accortezza, nell'eventuale aggiunta della relativa dependency al pom.xml del webadmin, **e solo se questa abbia come propria dependency webclient**, è quella di aggiungere solo le class:

1. pojo dei widget
2. pojo modello dati
3. XInitializerService.class che registra widget, tipi di menu e tipi di schede

Questo dovrebbe essere fatto in una delle modalità consentite da maven, per esempio

<code xml>

```
        <!-- issue #1257 -->
        <!-- dependency that have got gw-webclient as dependency HAVE TO
make a selection of resource to import -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <version>3.6.1</version>
            <executions>
                <execution>
                    <id>copy-gw-plugin-classes</id>
                    <phase>process-classes</phase>
                    <goals>
                        <goal>copy</goal>
                    </goals>
                    <configuration>
                        <artifactItems>
                            <artifactItem>
                                <groupId>com.geowebframework</groupId>
                                <artifactId>gw-additional-
plugin</artifactId>
                                <version>x.y.z</version>
<outputDirectory>${project.build.outputDirectory}/</outputDirectory>
<include>com/geowebframework/gw-additional-
plugin/service/xService.class</include>
                                <include>com/geowebframework/gw-additional-
plugin/model/*.class</include>
                                <include>com/geowebframework/gw-additional-
plugin/model/**/*.class</include>
                            </artifactItem>
                        </artifactItems>
                    </configuration>
                </execution>
            </executions>
        </plugin>
```

<code>