Creazione di un plugin per Geoweb

Geoweb è stato sviluppato in maniera da poter funzionare con vari plugin, che ne estendono le funzionalità di base. Tipicamente si crea un plugin per aggiungere al framework nuovi componenti, che possono essere:

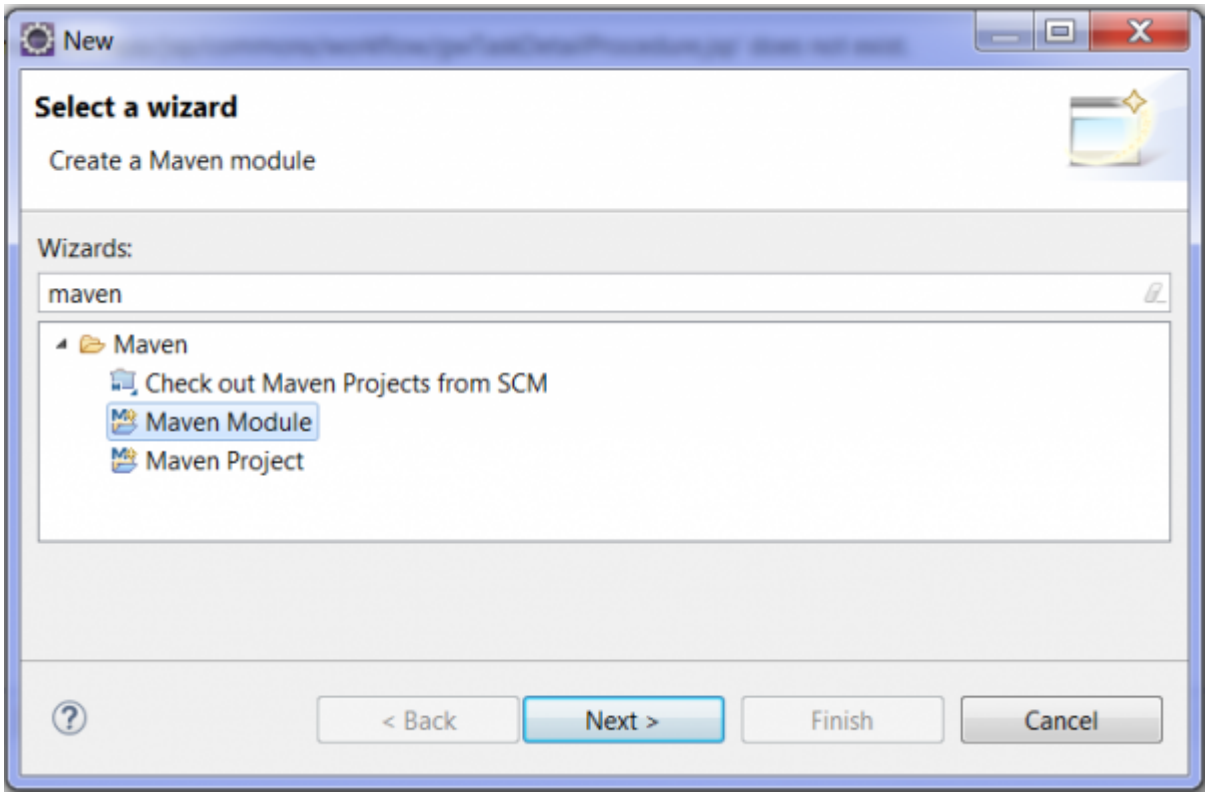
- schede
- widget
- leafItem
- java service lato server (servizi rest per comunicare con terzi, api accessibili da .groovy)
- function lato javascript (formatter lista, api js)

Per fare ciò, durante lo sviluppo, sono state stabilite convenzioni e sono stati creati meccanismi che permettono di poter produrre plugin per il framework, potenzialmente anche senza avere a disposizione il sorgente di Geoweb.

1. Creazione progetto plugin

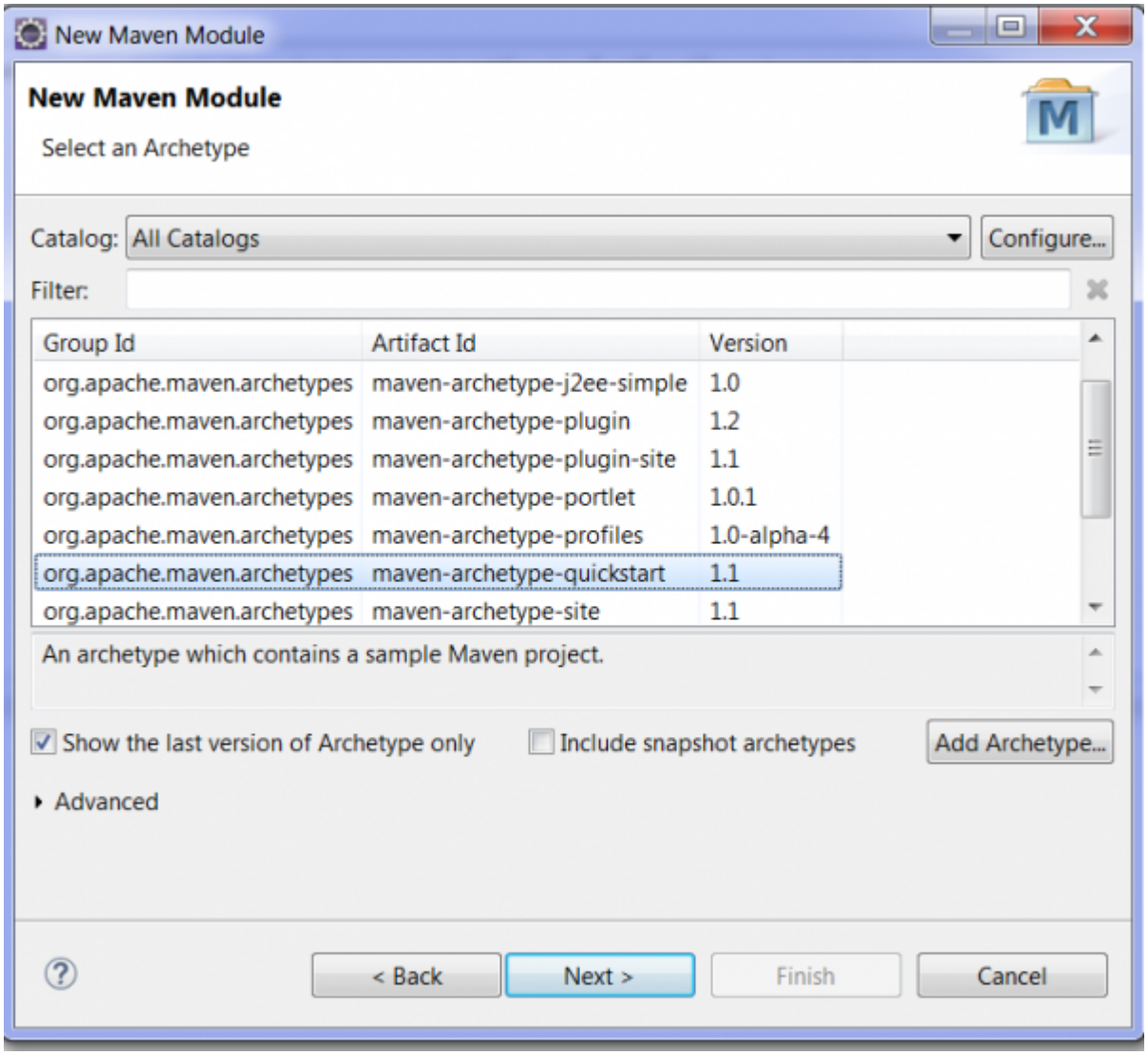
Mettiamo di voler creare il plugin *gwplugin*. I seguenti passi sono validi se si dispone del progetto maven completo di Geoweb.

- da dentro Eclipse: *File* ⇒ *New..* ⇒ *Maven Module*.

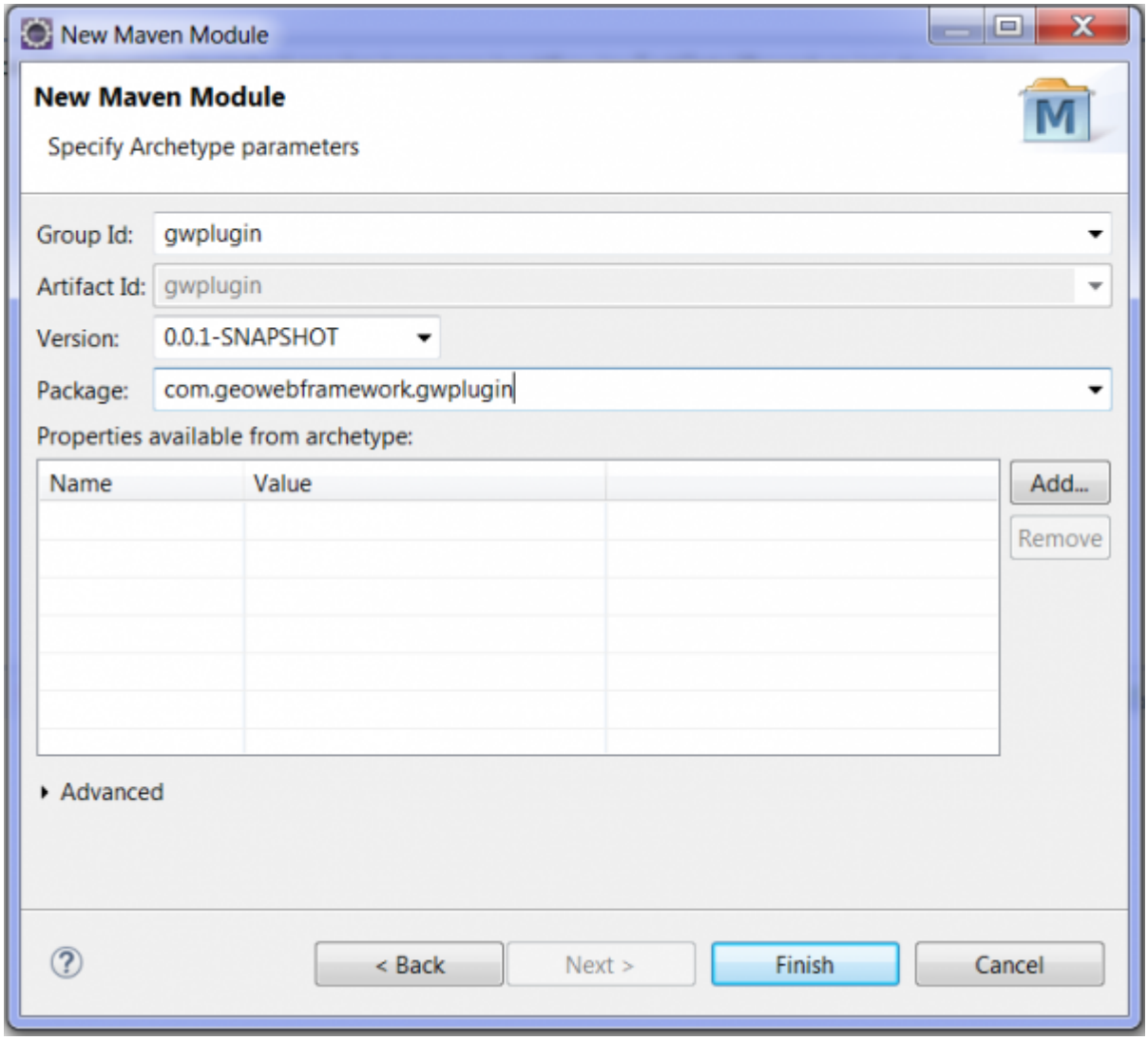


- Premere '*Next*'.
- si apre la scheda '*Select the module name and parent*'.
- inserire il nome del pugin per '*Module Name*'.
- selezionare 'geowebframework' in '*Parent Project*' .

- Premere '*Next*'.
- si apre ora la scheda '*Select an Archetype*'.



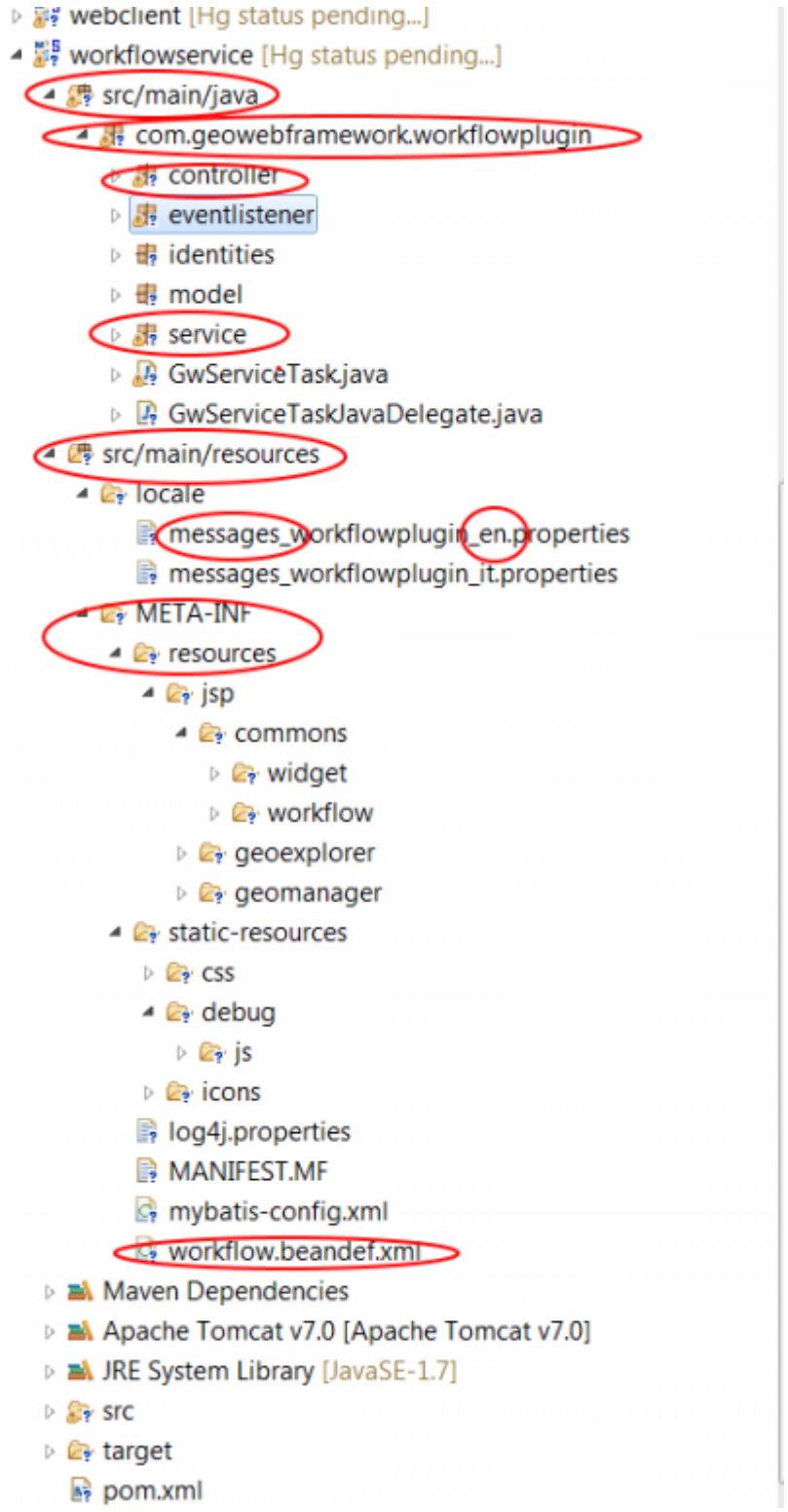
- selezionare l'archetype con '*Arttifact Id*' uguale a '*maven-archetiye-quickstart*'.
- Premere '*Next*'.
- Si apre la scheda '*Specify Archetype parameters*'



- in '*Group Id*' lasciare il nome suggerito
- in '*Package*' mettere per convenzione com.geowebframework.gwplugin
- Ora bisogna importare il progetto, File ⇒ Import.. ⇒ Existing Maven Projects
- selezionare la cartella base del framework 'c:\pathToEclispeWorkspace\GeoWebFramework' in 'Root Directory' e mettere la spunta sul progetto da importare in '*Projects*'
- Premere '*Finish*'.

2. Struttura progetto plugin

Questa è la struttura di massima per il progetto



La sezione *src/main/java* ospita il package che segue per convenzione il pattern 'com.geowebframework.' + nomepluginminuscolo Sempre per convenzione in genere vengono creati dei sotto package *controller, service* e *model* che ospitano le varie classi in base alla loro funzione.

La sezione *src/main/resources* ospita:

- **locale**: ospita i file di localizzazione. dato un locale, mettiamo en, la regola è che vengono inclusi in automatico tutti i file che stanno sotto la cartella locale/, che iniziano per '*messages_*' e finiscono per '*_en.properties*'. Per convenzione in mezzo si usa scrivere il nome del plugin.
- **META-INF**: per come è impostato il dispacher-servelt.xml del webclient essa deve contenere

- ***resources*** ospita i file jsp.
- ***static-resources*** ospita css, debug/js, icons
- ***xxx.beandef.xml*** nel dispacher-servelt.xml del webclient si definisce che in automatico vengano aggiunti alla configurazione tutti i bean spring dentro i file che matchano META-INF/*.beandef.xml .

```
     <?xml version="1.0" encoding="UTF-8"?>
     <beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mvc="http://www.springframework.org/schema/mvc"
     xmlns:context="http://www.springframework.org/schema/context"
     xmlns:mybatis="http://mybatis.org/schema/mybatis-spring"
xmlns:util="http://www.springframework.org/schema/util"
     xmlns:aop="http://www.springframework.org/schema/aop"
     xsi:schemaLocation="
             http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
             http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
             http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
             http://mybatis.org/schema/mybatis-spring
http://mybatis.org/schema/mybatis-spring.xsd
             http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd
             http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
         <context:component-scan base-package="com.geowebframework.gwplugin"
/>
         ...
         <bean id="pluginService"
class="com.geowebframework.gwplugin.service.pluginService" />
     </beans>
```

Nel ***pom.xml*** possono essere importate, tramite sintassi Maven, tutte le librerie java richieste. Dato che probabilmente il plugin è un maven module di progetto maven geowebframework, le librerie che stanno anche nel modulo principale possono anche importate omettendo la versione.

```
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd"
xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

     <modelVersion>4.0.0</modelVersion>

     <parent>
        <groupId>com.geowebframework</groupId>
        <artifactId>com.geowebframework</artifactId>
        <version>0.0.1-SNAPSHOT</version>
```

```
    </parent>

    <artifactId>gwplugin</artifactId>
    <name>gwplugin</name>
    <url>http://maven.apache.org</url>

    <properties>
   <maven.compiler.source>8</maven.compiler.source>
   <maven.compiler.target>8</maven.compiler.target>
       <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
       <!-- geoweb dependencies -->
       <dependency>
          <groupId>com.geowebframework</groupId>
          <artifactId>dataservice</artifactId>
          <version>${project.parent.version}</version>
       </dependency>
       <dependency>
          <groupId>com.geowebframework</groupId>
          <artifactId>metadataservice</artifactId>
          <version>${project.parent.version}</version>
       </dependency>
       <dependency>
          <groupId>com.geowebframework</groupId>
          <artifactId>transfer-objects</artifactId>
          <version>${project.parent.version}</version>
       </dependency>
       <!-- geoweb dependencies -->

       <dependency>
          <groupId>org.slf4j</groupId>
          <artifactId>slf4j-log4j12</artifactId> <!-- NO NEED TO SPECIFY
<version>1.7.7</version> -->
       </dependency>
    </dependencies>
</project>
```

3. InitializerService

Le dichiarazioni dei nuovi widget, leafItem, etc vengono in genere fatte tramite un'apposita classe di inizializzazione presente sotto il package **service** (che per convenzione si chiama come il plugin+'InitializerService', camel case. Es: *GwPluginInitializerService.java*)

Il *bean* di tale classe può essere dichiarato esplicitamente, o può essere istanziato in automatico tramite il component-scan. In entrambi i casi da dentro il ***gwplugin.beandef.xml***.

Questa è la struttura tipica della classe.

```
public class GwPluginInitializerService {
    private final static Logger log  =
Logger.getLogger(GwPluginInitializerService.class);

    @Autowired private GwRegistry gwRegistry;

    public GwPluginInitializerService(){
       super();
    }

    @PostConstruct
    public void init(){
          //registering widgets
          ...
          //registering leafItem
          ...
          //mapping plugin js resources
          ...
          //mapping plugin css resources
          ...
    }
}
```

Il bean della classe viene creato in avvio, e l'annotazione *@PostConstruct* causa l'esecuzione del metodo init() non appena tutti i bean dipendenti sono pronti ed istanziati.

Registrazione widget

Si rimanda all'apposita sezione per una descrizione approfondita / / TODO ADD LINK Qui da integrare c'è solo il fatto che eventuali risorse .js o .css usate dal widget possono essere incluse in modo agevole (vedi sotto).

Registrazione leafItem

Non vi sono oneri obbligatori. Un tipo di leafItem si dichiara, nell'xml di progetto, semplicemente usandolo nell'attributo type di un tag leafItem (posto sotto un tag leafItemContainer).

```
     <accordionPaneItem name="slm1" label="Second Level Menu Item" image=""
type="leafItemContainer">
         <leafItem name="tlm1" label="Thirdd Level Menu Item" image=""
type="gwPluginType">
         </leafItem>
     </accordionPaneItem>
```

Valgono poi tutte le istruzioni contenute nell'apposita sezione riguardante la definizione di nuove tipologie di Menu di Terzo Livello (LeafItem).

Dichiarazione risorsa javascript

I file js dichiarati da plugin saranno aggiunti all'apertura della pagina jsp principale del progetto, ma

per ultimi dopo le seguenti categorie di file javascript, in ordine:

1. js dojo framework
2. js customDojo
3. js comuni
4. js specifici del tipo di progetto.

L'ordine può non essere indifferente nel caso in cui nei file js del plugin si faccia riferimento ad entità javascript dichiarate altrove nel framework.

Le risorse verranno importate con usando un *tag* **<script>** con *attribute* **defer** (cioè che non verrà caricato finchè la pagina non sarà stessa non sara in stato *loaded*).

Il path da specificare parte dalla cartella **META-INF/static-resources**, ma possono essere anche inclusi file .js disponibili in rete scrivendo l'indirizzo internet per intero (partendo da 'http:/ / ')

```
	gwRegistry.addJsResource("debug/js/gwPlugin.js");

	gwRegistry.addJsResource("http://mysite.com/files/file.js");
```

Dichiarazione risorsa css

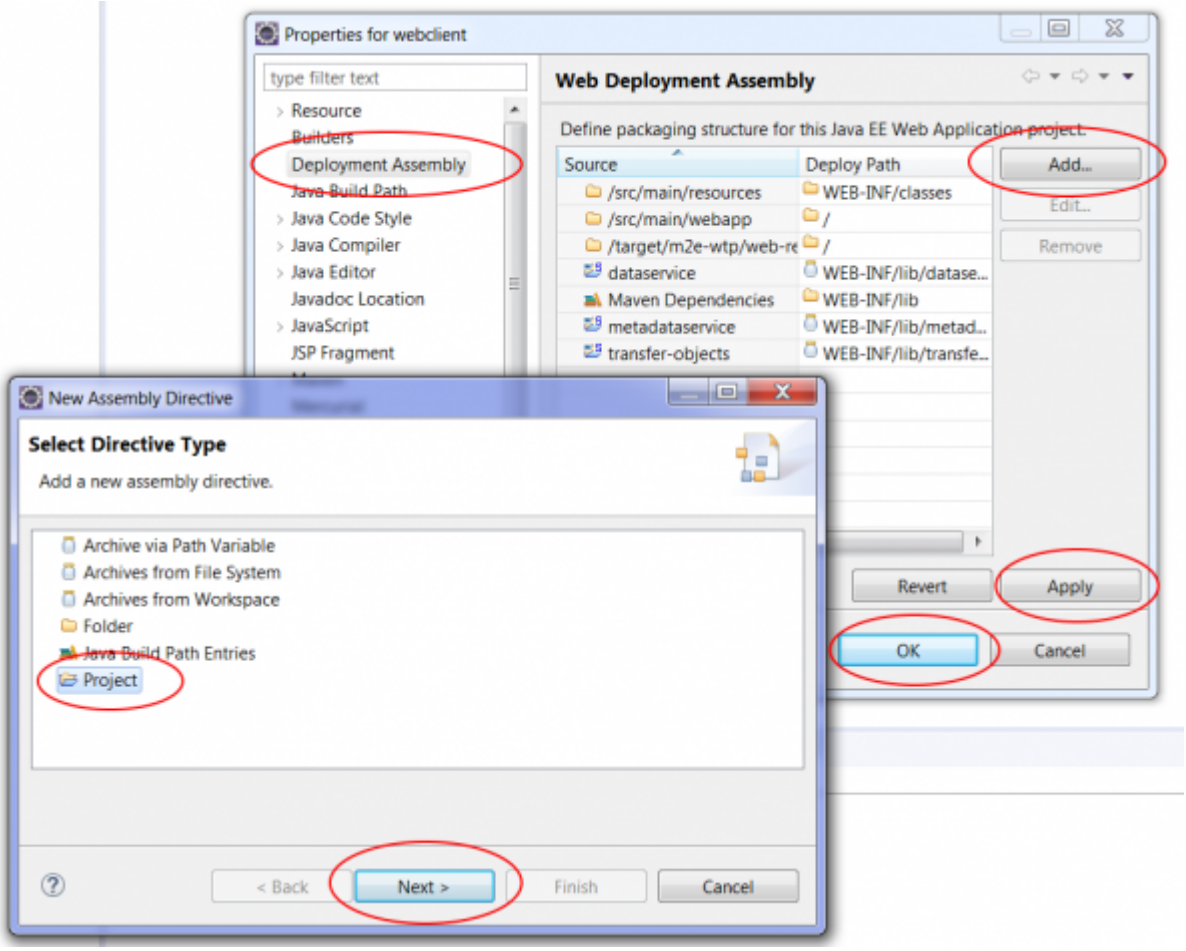
I file .css dichiarati da plugin saranno aggiunti all'apertura della pagina jsp principale del progetto, ma per ultimi dopo i css di dojo e quelli comuni. In generale ciò non ha alcuna conseguenza.

```
	gwRegistry.addCssResource("css/gwPlugin.css");
```

(DEPRECATO) (fino alla 4.2.14) Dispiegamento plugin

Un progetto plugin per Geoweb in fase di sviluppo può essere testato aggiungendolo alla webapp 'webclient', in questo modo:

- rClick su progetto 'webclient' ⇒ Properties..
- selezionare la scheda 'Deployement Assemby'
- click su Add
- Project ⇒ progetto_plugin ⇒ Ok
- Apply ⇒ Ok



Da notare che questa impostazione influisce sui plugin che verranno applicati a webclient sia in fase di run/debug dell'istanza del server Tomcat in locale. Ciò non toglie che eventuali setting per la compilazione del progetto all'interno di Eclipse vadano comunque sia impostati dentro 'Java Build Path'.

- Dentro scheda 'Project' importare i progetti contenenti classi usate nel plugin
- Dentro 'Libraries' assicurarsi di avere aggiunte
  - Java Libraries
  - Maven Dipendencies
  - Server Libraries

Tali setting influiscono anche in fase di build del .war duranteil processo esportazione.

In produzione, invece, un plugin risulta all'atto pratico un file .jar e per poter essere dispiegato necessita solo di essere posizionato sotto la cartella 'lib' della specifica cartella sotto '/webapps' del TomCat.