# WEB SERVICE NON SOLO PER L'APP

**Marco Ferrante**

Università di Genova
*marco@csita.unige.it*

***— WORKSHOP —***

*ARGOMENTO: Aspetti tecnici*

## Abstract

I Web Service esposti da Moodle, sviluppati prevalentemente per il funzionamento della App Mobile, possono essere sfruttati per l'integrazione di applicazioni e l'automazione di task amministrativi. Nel prosieguo del contributo sono illustrate le configurazioni di Moodle e i tool necessari per utilizzarli, come traccia di quanto realizzato durante il workshop.

**Keywords** – Web Services, integrazione

## 1 INTRODUZIONE

I Web Service (WS) sono API (Application Programming Interface) pensati per consentire l'interazione tra applicazioni tramite l'esecuzione di funzioni, disaccoppiando, tramite l'uso di protocolli standard, le modalità di comunicazione fra di esse. Moodle supporta differenti protocolli: SOAP, XML-RPC o REST; la scelta di quale protocollo conviene utilizzare dipende dalla natura e dalla specificità della applicazione che deve dialogare con Moodle e da quale supporto è presente nel linguaggio con cui è creata quest'ultima e la semplicità di utilizzo del protocollo.

Tipicamente applicazioni che siano sviluppate in java si troveranno più facilitate nell'uso di SOAP o XML-RPC, mentre applicazioni scritte in javascript sono facilitate nell'uso del protocollo REST/json e infine applicazioni scritte in php potranno indifferentemente utilizzare uno dei protocolli che Moodle consente di attivare.

Il workshop, di cui questo contributo è la traccia, ha il compito di illustrare le modalità con cui è possibile utilizzare i WS di Moodle per operazioni di integrazione e l'automazione di task amministrativi.

## 2 IL SETUP DI MOODLE

Per l'accesso da remoto via WS, Moodle espone un API con funzioni piuttosto differenti da quelle disponibili *in-process*, ad esempio per lo sviluppo di plugin. Le funzioni vengono organizzate in *servizi*, ognuno costituito da un nome, un insieme di funzioni esposte, un insieme di privilegi e un insieme di utenti autorizzati.

### 2.1 Impostazioni di base

Una descrizione passo-passo delle operazioni è disponibile nella pagina della propria installazione:

**Amministrazione del sito → Plugin → Web service → Panoramica**

Nel seguito, gli esempi saranno riferiti a Moodle 3.7 e verranno descritti solo i punti meno ovvi.

L'unico *servizio predefinito* disponibile di default è *Moodle mobile web service*, che non è adatto a compiti amministrativi perché mancano delle funzioni essenziali, come l'iscrizione di altri utenti (nel senso di "diverso da quello connesso") ad un corso, e in generale non si presta alle attività di amministrazione. I "servizi predefiniti" sono attivati dai plugin a livello di codice [7] e non possono essere modificati dall'amministratore.

Per attivare un nuovo servizio adatto all'integrazione con altri sistemi, selezionare:

**Amministrazione → Plugin → Web Services → Gestione servizi**

e compilare la scheda. Aprendo “Visualizza più elementi...” appaiono delle opzioni che permettono di abilitare le funzioni ottimizzate per download e upload e di richiedere un ruolo specifico per accedere al servizio. Ulteriori restrizioni possono essere poi associate ai *token* utente per l’accesso al servizio.

Al servizio vanno aggiunte le funzioni che deve esporre. Un elenco generico delle funzioni disponibili via web services si trova sul sito di documentazione di Moodle.org [8] ma è solo indicativo perché ogni singola installazione di Moodle può esporre funzioni diverse o addirittura usare parametri diversi per le stesse funzioni a seconda della configurazione specifica; l’architettura delle funzioni esportabili dai web services è concepita per renderle *auto-documentanti*.

Per avere l’elenco delle API effettivamente disponibili sul proprio server occorre abilitare da:

**Amministrazione → Plugin → Web service → Gestione protocolli**

la voce **“Documentazione web service”** e poi consultare (dando tempo al sistema di generare l’elenco completo, operazione piuttosto onerosa):

**Amministrazione → Plugin → Web service → Documentazione API**

Qui si troveranno le funzioni effettivamente disponibili in base alla versione installata, quelle esposte da plugin aggiuntivi e, soprattutto, i parametri per le funzioni standard che dipendono da altri plugin.

A questo scopo Moodle richiede che ogni modulo definisca le funzioni esportabili in uno specifico file sorgente, solitamente chiamato **externallib.php,** e che per ogni funzione API siano definite tre funzioni PHP: la funzione stessa *fun()*, che verrà esposta come *modulo_fun()*, e due di documentazione *modulo_fun_parameters()* e *modulo_fun_returns()*.

Uno sviluppatore può consultare questo file come estrema risorsa in caso di dubbi, ma deve tenere presente che i parametri potrebbero essere ricalcolati in base alla reale configurazione. Ad esempio, le funzioni del gruppo *core mod_assign_*()* relative ai compiti dipendono da come le attività possono essere valutate, ad esempio per la presenza di un sistema antiplagio, e quindi i parametri cambiano a seconda della configurazione. La differenza tra le due situazioni si può vedere in Tabella 1.

| | |
|---|---|
| Dalla pagina Documentazione web service | `assignmentid= int`<br>`plugindata[onlinetext_editor][text]= string`<br>`plugindata[onlinetext_editor][format]= int`<br>`plugindata[onlinetext_editor][itemid]= int`<br>`plugindata[files_filemanager]= int` |
| Da externallib.php | `$pluginsubmissionparams = array();`<br>`foreach ($instance->get_submission_plugins() as $plugin) {`<br>`  if ($plugin->is_visible()) {`<br>`  $pluginparams = $plugin->get_external_parameters();`<br>`  if (!empty($pluginparams)) {`<br>`  $pluginsubmissionparams =`<br>`  array_merge($pluginsubmissionparams,`<br>`  $pluginparams);`<br>`  }`<br>`  }`<br>`}`<br>`return new external_function_parameters(`<br>`  array(`<br>`  'assignmentid' => new external_value(PARAM_INT,`<br>`  'The assignment id to operate on'),`<br>`  'plugindata' => new external_single_structure(` |

| | |
|---|---|
| | $pluginsubmissionparams<br>)<br>)<br>); |

**Tabella 1 - Confronto tra codice documentazione e istanza del server**

## 2.2 Protocolli e privilegi utente

Come accennato in precedenza, Moodle supporta l'invocazione remota con i protocolli *XML-RPC*, *SOAP* e uno stile indicato come *REST*. Per un certo periodo è stato anche disponibile AMF, il protocollo RPC usato Adobe Flash, ma con il progressivo abbandono di quest'ultimo non ha più ragione di esistere.

XML-RPC [9] è fornito sino dalle prime versioni di Moodle perché alla base di Mnet (dominio di Single SingOn di Moodle/Mahara) seppure con accorgimenti tecnici custom come firma e cifratura in standard XML-Security [10].

L'implementazione di SOAP si basa sulla classe standard di PHP e quindi non produce dei WSDL adatti per generare gli *stub* in Java o C# [11].

Quello che viene indicato come "protocollo REST" in realtà ne è un'intepretazione in senso lato. Lo stile architetturale *RESTful* è chiaramente definito nella tesi di dottorato di Roy Fielding [12], ed ha tra i suoi punti caratteristici l'uso dei *verbi* HTTP per le operazioni CRUD e l'identificazione delle risorse come URL. Moodle usa i metodi GET e POST in modo intercambiabile, i parametri di input e la stessa funzione nella *query string* e restituisce identificatori locali. Il fatto che possa lavorare con JSON non è un tratto distintivo dello stile REST.

A dispetto di tutto questo, è il protocollo predefinito e meglio supportato. Gli esempi che seguono sono sviluppati per REST. Si consideri che ogni servizio è accessibile con tutti protocolli abilitati dal pannello:

**Amministrazione -> Plugin -> Web service → Gestione protocolli**

ma l'utente specifico deve avere il privilegio "usa web services <*protocollo*>" per poterlo utilizzare.

La corretta assegnazione dei privilegi è fondamentale; se errati possono prodursi situazioni in cui i risultati vengono restituiti senza errore ma incompleti o incoerente. Ad esempio, potrebbo essere restituiti solo gli utenti iscritti a certi corsi e non ad altri. Per funzioni di amministrazione è quindi preferibile definire un ruolo globale *ad-hoc* per l'utente dei web services in modo che i privilegi non dipendano dal contesto.

## 2.3 I token

Moodle usa il paradigma degli *access token* per autenticare le invocazioni dei web services. Il *token*, specifico per utente e per servizio, può essere generato in vari modi:

1. dall'amministratore dal pannello: **Amministrazione → Plugin → Web service → Gestione token** in cui è anche possibile restringere gli indirizzi IP validi per i client e impostare una scadenza del *token*
2. dall'utente stesso, se il servizio lo permette, aprendo: **Preferenze → Chiave di sicurezza** dove troverà le proprie chiavi e potrà eventualmente resettarle
3. sempre dall'utente, da l'URL: **https://your.moodle/login/token.php?username=<...>&password=<...>&service=<...>** da cui si ottiene il token in una struttura JSON. Occorre considere che questo sistema non funziona per utenti che usano sistemi di Sing SignOn con credenziali che non transitano da Moodle, come SAML 2.0 o Shibboleth, ma è l'ideale per applicazioni che usano un account locale di servizio.

Come detto, il *token* è specifico di un servizio, quindi non occorrono altri parametri per identificare il servizio stesso. I *token* generati autonomamente dall'utente non sono comunque mostrati agli amministratori.

# 3 PARAMETRI E FORMATO DATI

Moodle si aspetta che gli argomenti vengano passati alle funzion per nome; questo stile risulta fluente in linguaggi che lo prevedono come C# e Python, ma rende il codice un po' innaturale nei linguaggi che prevedono solo argomenti per posizione. Il *token* e la funzione invocata sono anch'essi parametri.

Poiché il "protocollo REST" usa indifferentemente i metodi HTTP POST e GET gli argomenti nella *query string*, per verifica le funzioni possono essere invocate anche con un comune browser o con un tool a riga di comando come **curl** [14]. Ad esempio, per una verifica rapida della configurazione, si può invocare con un browser la funzione **core_webservice_get_site_info()** che restituisce informazioni generali sul sito con questa struttura generale di URL:

**https://your.moodle/webservice/rest/server.php?wstoken=...&wsfunction=core_webservice_get_site_info**

Poiché la parte iniziale che comprende l'endpoint, il *token* e il nome della funzione:

**https://your.moodle/webservice/rest/server.php?wstoken=...&wsfunction=…**

è uguale per tutte le invocazioni, nel seguito verrà omessa. Per default risultato è serializzato in XML; se si preferisce il formato JSON, basta aggiungere il parametro **&moodlewsrestformat=json**.

La funzione **core_webservice_get_site_info()** fornisce anche un'informazione importante, cioè l'id dell'utente relativo al *token*, dato necessario per alcune funzioni.

Può esserci un piccolo problema per i siti con le *site policy* abilitate: al primo accesso (o ogni volta che le policy vengono modificate) viene chiesto di accettarle anche via web services, altrimenti tutte le chiamate produrranno un errore con descritto il problema. Occorre un accesso preliminare al sito Moodle oppure l'invocazione via web services della funzione **core_user_agree_site_policy()**.

L'invocazione via web services ha un *overhead* di avvio significativo, quindi a differenza delle funzioni di libreria quelle esportate prevedono di aggregare diverse richieste in una sola con parametri multipli. Ad esempio, la funzione per cercare un utente per indirizzo *uidnumber* (codice identificativo) o indirizzo mail ha come parametri la chiave di ricerca e i valori da cercare:

**...=core_user_get_users_by_field&field=username&values[0]=…**

**...=core_user_get_users_by_field&field=email&values[0]=demo@your.domain**

in modo da risolvere più utenti in un colpo solo passando un array di valori:

**...&values[0]=<indirizzo mail 1>&values[1]=<indirizzo mail 2>&…**

Il risultato è sempre un array di profili come in Listato 1:

```
<?xml version="1.0" encoding="UTF-8" ?>
<response>
<multiple>
<single>
<key name="id"><value>612</value>
</key>
<key name="username"><value>xxxxxx</value>
</key>
<key name="firstname"><value>...</value>
</key>
<key name="lastname"><value>...</value>
...
</key></single></multiple></response>
```

**Listato 1 – Risposta a core_user_get_users_by_field()**

Si presti attenzione che i risultati non sono necessariamente nello stesso ordine dei parametri.

Il server potrebbe avere dei limiti alle dimensioni dell'URL o della *query string*; ad esempio Apache *httpd* ammette di default 8190 byte di lunghezza massima di una riga [15]. In questi casi è utile segnalare che le tutte chiamate possono essere anche POST con i parametri nel body codificati **application/x-www-form-urlencoded**.

# 4 GESTIONE DEGLI ERRORI

La filosofia dei web services di Moodle è estesa alla gestione degli errori; errori di protocollo vengono gestiti con normali stati HTTP (es. 403 per Autorizzazione negata a livello di rete), mentre errori di invocazione delle API restituiscono uno stato 200 con un oggetto contenente l'errore (ad esempio, autorizzazione negata per indirizzo IP non permesso dal *token*).

```
<EXCEPTION class="webservice_access_exception">
<ERRORCODE>accessexception</ERRORCODE>
<MESSAGE>Eccezione nel controllo accesso</MESSAGE>
<DEBUGINFO>Invalid service - IP:10.186.20.30 is not
supported - check this allowed user</DEBUGINFO>
</EXCEPTION>
```

**Listato 2 – Risposta XML di errore**

# 5 UPLOAD E DOWNLOAD DI FILE

Varie funzioni prevedono l'*upload* di file, ad esempio **core_files_upload()**, ma il contenuto va codificato in base64, con un aumento di dimensioni di circa il 30% e un significativo sovraccarico computazionale. Sono quindi disponibili due endpoint specifici per il *download* e l'*upload* di file [16].

L'upload avviene con un chiamata POST con *payload* i campi con nome, percorso e contenuto del file. In caso di successo, viene restituito un ***id*** del file, che si trova caricato in nell'area *draft* non visibile dalla propria interfaccia web, che nella forma … può essere utilizzato come argomento di altre funzioni, come **mod_assign_save_submission()**.

Fare attenzione che il demo ufficiale di Moodle [19] per PHP mostra:

**$params = array('file_box' => "@".$imagepath,'filepath' => $filepath, 'token' => $token);**

ma il parametro nella forma "**@**" è deprecato da PHP 5.6 e non più supportato da PHP 7 [17]. Usare invece la forma con **'file_box' => new \CURLFile($imagepath)** perché quella vecchia fallisce, inviando un contenuto nullo, senza dare alcun tipo di errore.

## Risoluzioni degli identificatori

Le operazioni che hanno per oggetti entità di Moodle come utenti, corsi, categorie, ecc.. richiedono il relativo identificatore *id* interno al sistema. Per integrare il sito Moodle con altri sistemi, è spesso necessario un'operazione preliminare di risoluzione degli identificatori esterni. Un esempio è già stato riportato con la funzione **core_user_get_users_by_field()** in per estrarre l'***id*** Moodle di un utente a partire dall'indirizzo email o dalla matricola se mappata su *idnumber*. Di solito è presente anche una simmetrica operazione di ricerca su campi multipli, ad esempio per gli utenti **core_user_get_users()** che a fronte di maggior versatilità ha prestazioni inferiori e non garantisce l'univocità delle risposte.

La stessa situazione duale si ritrova per i corsi con **core_course_get_courses_by_field()** e **core_course_get_courses()** e per tutte le altre entità.

L'***id*** utente interno è poi usato per tutte le funzioni relative ad essi. Ad esempio per inviare un messaggio privati, in questo caso passandone gli argomenti in POST, il parametro ***touserid*** identifica il destinatario del messaggio:

**$ curl -d "messages[0][touserid]=730&messages[0][text]=Testo messaggio" -H "Content-Type: application/x-www-form-urlencoded" -X POST "...=core_message_send_instant_messages"**

```
[{"msgid":6,"text":"<p>Testo
messaggio<\/p>","timecreated":1570549908,"conversationid"
:5,"useridfrom":730,"candeletemessagesforallusers":false}
]
```

**Listato 3 – Risposta JSON all'invio core_message_send_instant_messages()**

Come si vede in **listato 3**, il messaggio è salvato come frammento HTML. Esattamente come se fosse inviato dal sito, il messaggio verrà anche recapitato per posta elettronica.

# 6 SVILUPPO CON I WEB SERVICES DI MOODLE

Ogni plugin può esporre dei servizi remoti. La lettura della relativa documentazione [18] può risultare comunque utile anche agli sviluppatori di client web services.

Lo sviluppo di integrazioni lato client è ovviamente facilitato dall'uso di librerie già pronte; Moodle.org offre del codice dimostrativo in vari linguaggi [19].

## 6.1 Librerie PHP

L'aspetto più complesso dell'accesso via PHP è la serializzazione dei parametri, che per supportare così tanti protocolli risulta un poco controintuitiva. Ad esempio, una funzione come **enrol_manual_enrol_users()** a cui sarebbe sufficiente come parametro concreto un array di terne [ruolo, utente, corso], invece che formalizzarlo come ci si potrebbe aspettare come [[r1, u1, c1], [r2, u2, c2], …] richiede invece un array con unico elemento con chiave 'enrolments':

**$param = ['enrolments' => [['roleid' => .., 'userid' => ..., 'courseid' => ...], ['roleid' => ..., …]]**

Nel demo fornito da Moodle, la serializzazione è affidata a due funzioni specifiche, **format_array_postdata_for_curlcall()** e **format_postdata_for_curlcall()**, che possono essere usate come prototipo considerando che sono rilasciate sotto licenza GPL General Public License.

L'uso del demo ufficiale è però poco fluente; una libreria più in stile PHP è quella di Lawrence Lagerlof [20] che permette di scrivere :

**$MoodleRest = new MoodleRest('https://your.moodle/webservice/rest/server.php', $token);**

**$groups = $MoodleRest->request('core_group_get_groups', array('groupids' => array(1,2)));**

In UniGe è stata sviluppata una libreria che sfruttando i *metodi magici* [21], per cui l'oggetto **proxy** espone le funzioni remote come metodi locali:

**$moodle = new \UniGe\MoodleWSClient('https://yourmoodle/', $token);**

**$site_info = $moodle->core_webservice_get_site_info();**

**echo "Sei collegato al sito {$site_info->sitename}\n";**

## 6.2 Altri linguaggi

I demo di Moodle.org [19] comprendono codice JavaScript basato su jQuery [22] e quindi pensato per l'uso in un browser. Per l'uso lato server per *node.js* è disponibile il modulo di terze parti [23].

Gli esempi in Java e C# forniti da Moodle.org [19] risultano macchinosi perché REST non è il protocollo "naturale" per l'RPC in Java e come detto SOAP non risulta facilmente utilizzabile.

Per Perl, oltre il demo. è anche disponibile l'elegante libreria [24], purtroppo non molto aggiornata.

Python non è stato preso in considerazione negli esempi ufficiali, ma esiste un progetto specifico [25].

## Riferimenti bibliografici

[7] Moodle docs, *Adding a web service to a plugin*, https://docs.moodle.org/dev/Adding_a_web_service_to_a_plugin

[8] *Web service API functions*, https://docs.moodle.org/dev/Web_service_API_functions

[9] *XML-RPC Specification*, http://xmlrpc.scripting.com, (1999)

[10] *XML Security Specifications*, https://www.w3.org/2007/xmlsec/

[11] Moodle docs, *Does the Moodle SOAP server work with JAVA or .NET?*, https://docs.moodle.org/37/en/Web_services_FAQ#Does_the_Moodle_SOAP_server_work_with_JAVA_or_.NET.3F

[12] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, (2000).

[13] Roy Thomas Fielding *et alt*., *Hypertext Transfer Protocol – HTTP/1.1*, IETF RfC 2616

[14] Daniel Stenberg, *Everything curl*, ISBN 978-91-639-6501-2, (2018), disponibile con il software in https://curl.haxx.se/

[15] Apache httpd, *LimitRequestLine Directive*, http://httpd.apache.org/docs/2.4/mod/core.html

[16] *Web services files handling*, https://docs.moodle.org/dev/Web_services_files_handling

[17] PHP manual, *curl_setopt*, https://www.php.net/manual/en/function.curl-setopt.php

[18] Moodle docs, *Adding a web service to a plugin*, https://docs.moodle.org/dev/Adding_a_web_service_to_a_plugin

[19] *Demo web service clients for Moodle 2*, https://github.com/moodlehq/sample-ws-clients

[20] Lawrence Lagerlof, *MoodleRest*, https://github.com/llagerlof/MoodleRest

[21] PHP manual, *Magic Methods*, https://www.php.net/manual/en/language.oop5.magic.php

[22] *jQuery*, https://jquery.com

[23] David Mudrák, https://github.com/mudrd8mz/node-moodle-client

[24] Andrew Solomon, https://metacpan.org/pod/WebService::Moodle::Simple

[25] https://pypi.org/project/moodle/