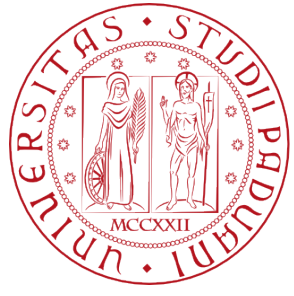
Creazione e Gestione di una API REST

Diego Casella
February 24, 2015



Università degli Studi di Padova

Contenuti

Introduzione

In questo sesto seminario affronteremo in maniera breve, ma concisa, come realizzare un servizio REST attraverso il quale rendere fruibili dei contenuti presenti in un nostro sito.
Enuncieremo inoltre quali siano le *best practices* da adottare e, per comoditá, ci appoggieremo al framework Drupal.

Cos' é REST

Col termine *REST* si intende una tipologia di architettura software che permette di eseguire operazioni in un server remoto attraverso l'uso di chiamate HTTP standard. L'acronimo deriva da *Representational State Transfer*, ovvero da una architettura software che definisce come le risorse web possano essere definite ed indirizzate con dei semplici url e metodi HTTP, attraverso delle precise e definite specifiche che possono essere interpretate dal server.

Cos' é REST - Principi

L'architettura REST si fonda sui seguenti principi chiave di progettazione:

- un protocollo, che é di tipo client-server, stateless, ed a livelli;
- ogni risorsa é indirizzabile attraverso un URL univoco;
- ogni richesta non puó fare riferimento a piú risorse contemporaneamente;
- tutte le risorse sono condivise tra client e server attraverso un'interfaccia (client dá un comando e il server torna l'oggetto corrispondente), che consiste in:
  - per il comando, un insieme vincolato di operazioni ben definite;
  - per la risorsa, un insieme vincolato di contenuti;

Cos' é REST - Esempi

Supponiamo che, ad esempio, *http://math.unipd.it* fornisca una interfaccia REST per accedere ai docenti afferenti al dipartimento di Matematica, per ognuno dei quali sia possibile accedere al numero di telefono del suo ufficio ed email.

Esempio - URL per eseguire una query dei docenti presenti

http://math.unipd.it/professors (HTTP method: GET)

Cos' é REST - Esempi (cont.)

Dopo aver individuato il docente interessato, rappresentato da un proprio *id* univoco, si puó richiedere il suo numero di telefono o email attraverso le seguenti chiamate:

Esempio - Reperimento del numero di telefono

http://math.unipd.it/professors/<id>/phone (HTTP method: GET)

Esempio - Reperimento dell'email

http://math.unipd.it/professors/<id>/email (HTTP method: GET)

Cos' é REST - Metodi HTTP

Gli esempi visti fino ad ora hanno sempre interessato dei metodi HTTP di tipo GET, tuttavia esso non é l'unico metodo disponibile. Esistono infatti piú metodi a diposizione, quelli piú utilizzati sono:

- *GET*, che come visto in precedenza serve ad ottenere una risorsa descritta dall'URL specificato;
- *POST*, per modificare ed aggiornare una risorsa nel server;
- *PUT*, per creare una nuova risorsa nel server;
- *DELETE*, per eliminare una risorsa presente nel server;

Cos' é REST - Metodi HTTP (cont.)

Sebbene POST e PUT possano sembrare due metodi intercambiabili fra di loro, e difatti molti servizi REST li confondono, é buona prassi utilizzarli tenendo presente che:

- *PUT* viene usato per creare o sovrascrivere una risorsa. É un operatore idempotente, e quindi eseguire un PUT piú volte consecutive non produce altrettante copie della risorsa;
- *POST* invece viene usato per modificare una risorsa, dunque se tale risorsa non esiste viene generato un errore;

Cos' é REST - Risorse

Le risorse che si vogliono ricevere o inviare al server, come detto in precedenza, devono avere una struttura ben definita nei contenuti, e vengono descritte tipicamente attraverso un formato di tipo XML oppure JSON, entrambi molto adatti al parsing ed al riconoscimento dei campi che li costituiscono.

Implementazione di REST in Drupal

Dopo aver visto brevemente come é costituito un servizio REST, ci occuperemo di come implementarlo ed inserirlo in un sito web pre-esistente. É pensabile di realizzarlo in maniera completamente slegata dal sito stesso, ma questa rappresenta una scelta azzardata e limitante, e soprattutto difficilmente mantenibile.
Rifacendoci all'esempio del servizio REST per accedere ai docenti del dipartimento di Matematica, si supponga di aver scritto tutto il codice PHP che esegua le query al database dei docenti, delle mail e dei numeri di telefoni ad essi associati. Cosa succede se un aggiornamento al database cambiasse, per ragioni di ottimizzazione/performance, la disposizione delle tabelle, o come le varie colonne vengono identificate? Saremmo costretti a rivedere il codice PHP che esegue tale query, con complessitá crescente col crescere delle proprietá presenti in una specifica risorsa.

Implementazione di REST in Drupal - 2

Dunque, per ragioni di manutenibilitá, é piú saggio appoggiarsi ad un framework esistente, che semplifichi e si prenda carico di fornire una astrazione di alto livello per l'accesso ai dati presenti nel sito. Drupal in tal senso é un framework molto flessibile ed adatto allo scopo essendo modulare, estensibile e documentato.
Grazie a Drupal é possibile infatti implementare un servizio REST che sia realizzato come un modulo Drupal stesso, dando la possibilitá di renderlo dipendente da altri moduli in esso presente o di terze parti, e di fornire una interfaccia web per il suo stesso monitoraggio e gestione.

Modulo Rest Drupal

Una volta decisa la struttura delle url che interessano le chiamate REST, e le risorse che verrano scambiate fra client e server, si puó partire con l'implementazione del modulo Drupal che realizzerá il servizio REST.
La prima azione da intraprendere é il riconoscimento ed il parsing del'URL, fasi necessarie per interpretare l'operazione che l'utente vuole eseguire.

Modulo Rest Drupal - hook_menu()

Giá in questa fase preliminare si apprezza l'utilizzo di un framework quale Drupal, per riconoscere ed estrarre informazioni dall'URL fornito.
Difatti la callback hook_menu(), reimplementata dal progettista del modulo rest, permette di registrare degli URL, estrarre informazioni da esso, ed associarvi una corrispondente funzione che verrá invocata quando tale URL verrá interrogato.

Modulo Rest Drupal - hook_menu()

Esempio di hook_menu()

```
function my_custom_module_menu() {
    $urls['professors/%/phone'] = array(
        'page callback' => 'get_phone_number',
        'page arguments' => array(1),
    );
    return $urls;
}
function get_phone_number($args) {
    # ensure the correct HTTP method
    # do something with the id passed in args array
    echo my_format( $phone_number ); # in JSON or XML format
}
```

Modulo Rest Drupal - Pagina di Gestione

Durante il ciclo di vita del servizio REST, spesso accade che si debba aggiornare o fermare tale servizio, per poter effettuare una manutenzione/backup del database, o un upgrade/estensione dell'API.
Ció comporta che il servizio REST vada fermato, ed in tal caso cosa succede dal punto di vista dell'utente utilizzatore del nostro servizio?
Di punto in bianco ovviamente, l'applicazione che sfrutta il nostro servizio REST smette di funzionare senza alcuna ragione apparente, quindi occorre pianificare anzitempo delle soluzioni per poter mantenere informato l'utente.

Modulo Rest Drupal - Pagina di Gestione - 2

Ancora una volta, Drupal si dimostra utile nella semplificazione di questo compito: é possibile infatti realizzare una pagina di configurazione, accessibile dall'amministratore del sito, che possa ad esempio permettere di impostare il server REST in due modalitá differenti:

- *online* quando il servizio é operazionale;
- *maintenance* quando il servizio é sotto manutenzione/aggiornamento;

Cosıfacendo, l'utente é costantemente a conoscenza dello stato del servizio.

REST best practices - Docs

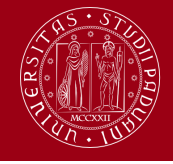
REST é anzitutto una interfaccia attraverso la quale uno sviluppatore accede ad un servizio: una buona API REST *deve* essere corredata di una buona documentazione, tale da:

- descrivere la funzionalitá di ogni interfaccia, metodo e risorsa che é possibile che appare nell'API pubblica;
- corredare quanto piú possibile la documentazione di codice di esempio;
- fornire una o piú applicazioni di esempio che consentano di realizzare un eseguibile, per quanto minimale, tale da poter essere preso in esempio ed essere esteso da altri sviluppatori esterni;

REST best practices - API Versioning

Quando si crea e gestisce una qualsiasi API, prima o poi ci si viene a trovare di fronte alla scelta di dover rivedere e talvolta riscrivere porzioni piú o meno grandi di una o piú interfacce presenti. Per questa ragione, é consono utilizzare un sistema di versioning composto da due o piú numeri del tipo *v1.2.3*.

REST best practices - API Versioning (2)



Nell'esempio precedente, *v1.2.3*:

- "1" indica la *major version*, cioé la versione dell'API stessa. A versioni differenti corrispondono API spesso differenti fra loro quindi(classi/metodi rinominati o eliminati, behavior di esse differenti), se si volesse aggiornare una applicazione per fare uso di una versione piú recente di una data API, si deve tener conto di un lavoro consistente per adeguare il codice;
- "2" indica la *minor version*, ovvero cambiamenti avvenuti nell'API che peró sono retrocompatibili, come ad esempio aggiunta di nuovi metodi o classi;
- "3" indica la *patch version*, ed ún numero che viene incrementato a seconda delle patch e bug fixes introdotti;

REST best practices - API Versioning (3)

Ora che abbiamo visto l'utilitá dell'API versioning, é opportuno fare in modo che tale stringa sia integrata nell'URL. Cosí facendo infatti, si rende l'API stessa esplorabile tramite un browser web (per quanto possibile), facilitando il lavoro dello sviluppatore, soprattutto agli inizi, oltre ovviamente al fatto di rendere obbligatorio specificare quale API version adottare.

REST best practices - Accesso risorse

Utilizzare *nomi*, non *verbi*, per specificare quale collezione di risorse accedere.

Esempio

http://math.unipd.it/getprofessors # sbagliato
http://math.unipd.it/professors # corretto

Questo perché é tramite il tipo di metodo HTTP (GET, POST, PUT, DELETE) che andremo ad istruire il server su quale azione intraprendere in una determinata risorsa, non attraverso l'URL!

REST best practices - Accesso risorse (2)

Rendere ogni elemento di una collezione indirizzabile tramite *id* univoco, mentre le sue proprietá tramite nome. Se una proprietá é a sua volta una collezione di risorse, le sue sottorisorse vanno anch'esse riferite tramite *id*.

Esempio

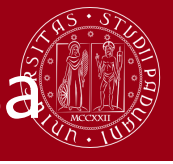
http://math.unipd.it/professors/143/mail
http://math.unipd.it/professors/143/pubblications/5

REST best practices - Complessitá

Con l'aumentare del numero delle risorse cresce, in maniera proporzionale, la lunghezza dell'URL necessaria per eseguire una query. É facile intuire che in situazioni reali, questo si traduce in URL troppo estese: é necessario dunque suddividere tale complessitá fra URL e risorse scambiate col server.
Questo é fondamentale soprattutto perché, in linea di principio, non si vuole creare un ingorgo nel server dovuto a troppe richieste a proprietá o risorse dettagliate: meglio ritornare piuttosto una risorsa piú consistente, che poi il client elaborerá separatamente.

REST best practices - Ritornare sempre una risorsa

Nel caso l'HTTP request sia di tipo GET, questo é ovvio. Meno intuitivo é il caso dei metodi POST, PUT, e DELETE. Dal momento che comunque questi metodi cambiano una risorsa presente nel server, questo comporta che l'utilizzatore dell'API voglia anche sapere cosa é successo alla risorsa interessata *dopo* la sua modifica. Quindi é preferibile evitare di fargli eseguire due richieste HTTP differenti. Semplicemente, si faccia in modo da ritornare la risorsa aggiornata, e sará poi a discrezione del programmatore tenerne conto o meno.



Questo é intuitivo, in quanto dobbiamo fornire operazioni per molteplici valori, e anche per il singolo valore.

Esempio - GET (lettura)

GET *http://math.unipd.it/professors* # ritorna tutti i professori
GET *http://math.unipd.it/professors/143* # ritorna uno specifico

Esempio - POST (inserzione)

POST *http://math.unipd.it/professors* # inserisce un nuovo professore
POST *http://math.unipd.it/professors/143* # Errore, l'id é generato dal server, noi non possiamo dire di creare una risorsa con un determinato id

Nota: tenere a mente che la risorsa dove sta operando il metodo HTTP é *professors*.

REST best practices - Due URL per risorsa

Esempio - PUT (update)

PUT *http://math.unipd.it/professors* # aggiorna la risorsa *professors* in blocco
PUT *http://math.unipd.it/professors/143* # aggiorna la risorsa con id=143, altrimenti dá errore: questo perché l'inserzione é adibita al metodo POST.

Esempio - DELETE (rimozione)

DELETE *http://math.unipd.it/professors/* # rumuove tutti i professori
DELETE *http://math.unipd.it/professors/143 # rimuove il professore con id=143*

REST best practices - Errori

Nel caso si verifichi un errore, é necessario fornire adeguate informazioni all'utilizzatore dell'API. Solitamente é consuetudine utilizzare codici di errore che siano compatibili con lo standard HTTP (200 - Ok, 404 - Not Found ecc..), comprensivi inoltre di un internal error code specifico per il servizio utilizzato, una stringa che dia maggiori informazioni sul tipo di errore. Inoltre sarebbe preferibile predisporre un campo che contenga un link alla pagina web dell'errore corrispondente, per cercare di guidare l'utente finale ad una risoluzione del problema.

REST best practices - JSON vs XML

Altra considerazione da fare riguarda il tipo di formato col quale esprimere le risorse scambiate tramite REST API. XML é stato per anni lo standard de facto per REST, tuttavia tale linguaggio é ridondante, di difficile parsing, e la sua capacitá di essere esteso é completamente ininfluente in un contesto dove l'unico scopo é quello di serializzare informazioni contenute in un server al mondo esterno.
Si preferisce quindi utilizzare lo schema JSON, che invece é piú semplice da utilizzare e elaborare.

Conclusioni

Riferimenti

- *Representational State Transfer*, `http://en.wikipedia.org/wiki/Representational_State_Transfer` ;
- *Best Practices for Designing a Pragmatic RESTful API*, `www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api` ;
- *5 Best Practices for Better RESTful API Development*, `http://devproconnections.com/web-development/restful-api-development-best-practices` ;

Q&A