mer 10 febbraio 2021 - Lo Sviluppatore anno VI
Lo Sviluppatore

RESTful API: 10 Best Practices

Design by *Giuseppe Capodieci* -

Condividi

Negli ultimi anni lo sviluppo di web API si è notevolmente incrementato a seguito della diffusione di client etereogenei e di nuove tecnologie: PC desktop, dispositivi mobili come cellulari e tablet, IOT (Internet of Things), Big Data, usano tutti interfacce web based per connettere e aggregare componenti e dati distribuiti al fine di creare sempre nuove soluzioni/servizi per qualsisi tipo di business globale. E' nata per questo la necessità di creare dei sistemi di backend "neutrali" che forniscono dati in maniera "grezza" (tipicamente nel formato JSON o XML) che poi vengono utilizzati opportunamente sulle varie tipologie di dispositivi. Il modello adottato per la progettazione delle API è l 'ormai conslidato REST.

In breve, questo modello consente di strutturare le API in risorse logiche su cui si opera mediante il protocollo HTPP usando le operazioni standard che quest'ultimo ci mette a disposizione, quindi: GET, POST, PUT e DELETE.

Esula da questo articolo l'approfondire il modello REST; fornirò alla fine un elenco di risorse utili per chi volesse approfondire l'argomento. Qui invece verrano descritte 10 best pactices per progettare/sviluppare una API in maniera chiara e pulita. L'utilizzo di metodologie e convenzioni chiare e predefinite consente meglio di condividere le informazioni tra sviluppatori, architetti e designer al fine di migliorare tutto il ciclo di implementazione di una web API.

Ma vediamo l'elenco delle 10 best practices:

Indice

mer 10 febbraio 2021 - Lo Sviluppatore anno VI

1. Usare Nomi e non Verbi

Per una maggiore chiarezza usa il seguente schema per tutte le risorse:

Risorsa	GET read	POST create	PUT update	DELETE
/books	Ritorna una lista di libri	Crea un nuovo libro	Aggiorna i dati di tutti i libri	Elimina tutti i libri
/books/145	Ritorna uno specifico libro	metodo non consentito (405)	Aggiorna uno specifico libro	Elimina uno specifico libro

Non usare verbi:

```
/getAllBooks
/createNewBook
/deleteAllThrillerBooks
```

ad esempio, non sono buone soluzioni.

2. Usare i Nomi al plurale

Non utilizzare un mix di nomi al singolare e al plurare. Semplifica usando solo i nomi al plurale per tutte le risorse.

```
/books invece di /book
/users invece di /user
/products invece di /product
/authors invece di /author
```

3. GET e parametri di query non dovrebbero alterare lo stato

Usa i metodi **PUT, POST** e **DELETE** per alterare lo stato di una risorsa.

Non usare il metodo **GET** per i cambiamenti di stato:

```
GET /users/714?activate oppure
GET /users/714/activate
```

4. Usa le sub-resources per descrivere le relazioni

Se una risorsa è legata ad un'altra risorsa usa le sub-resources:

mer 10 febbraio 2021 - Lo Sviluppatore anno VI

```
GET /books/411/authors/ Restituisce la lista degli autori del libro 411
GET /books/411/authors/1 Restituisce l'autore #1 del libro 411
```

5. Usa gli header HTTP per definire i formati di serializzazione dei dati

Entrambi, client e server, hanno bisogno di sapere quale formato viene utilizzato per la comunicazione. Il formato deve essere specificato nel header HTTP della request. In particolare usando:

Content-Type : definisce il formato della request.
Accept : definisce un elenco dei formati di risposta accettabili.

6. Use HATEOAS

Hypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate è un principo secondo il quale gli hypertext links dovrebbero essere utilizzati per una migliore navigazione tra le risorse della API.

```
{
  "id": 411,
  "title": "La metamorfosi",
  "editor": "RCS",
  "ISBN": "9784563753",
  "pages" : 200,
  "authors": [
   {
    "id": "1",
    "name": "Franz Kafka",
    "links": [
     {
     "rel": "self",
     "href": "/api/v1/authors/1"
     }
    ]
   }
  ]
}
```

7. Implementa operazioni di filtraggio, ordinamento, selezione di specifici campi e paginazione per le collection

Filtraggio

```
GET /books?author=Franz+Kafka Ritorna la lista dei libri scritti da Kafka
GET /books?pages<=200 Ritorna una lista di libri che hanno al max 200 pagine
```

Ordinamento

Consenti l'ordinamento in base a uno o più campi.

```
GET /books?sort=-pages,+author
```

Questo ritorna un elenco di libri ordinati per numero di pagine in maniera discendente e per autore in maniera ascendente.

Selezione di campi

I dispositivi mobili visualizzano un insieme ristretto di dati per evidenti motivi di spazio nello schermo, quindi in genere non hanno bisogno di tutti i dati di una risorsa. E' bene quindi consentire al consumatore dell'API una selezione dei soli campi di una risorsa che interessano. Questo oltre a rendere più "duttile" l'API porta ad una riduzione del traffico di rete.

```
GET /books?fields=title,author,id
```

mer 10 febbraio 2021 - Lo Sviluppatore anno VI

Paginazione

Implementa la paginazione dei dati mediante l'uso dei parametri *limit* e *offset*. Questo è un meccanismo flessibile per l'utente ed è tipicamente utilizzato con i dati provenienti da un DB. E' bene impostare dei valori di default per questi parametri qualora questi non vengano specificati nei parametri della URL. Ad esempio *limit = 20* e *offset = 0*

Per richiedere l'invio di tutte le entità di un certo tipo utilizzare l'header HTTP non standard : *X-Total-Count*.

I link alle pagine precedente e successiva dovrebbero essere forniti nell'header HTTP *Link*. E' sempre meglio seguire questi link, indicati come valori di *Link*, per implementare la paginazione invece di costruirsi gli URL da se.

```
Link: <https://www.myhost.com/sample/api/v1/books?offset=15&limit=5>; rel="next",
<https://www.myhost.com/sample/api/v1/books?offset=50&limit=5>; rel="last",
<https://www.myhost.com/sample/api/v1/books?offset=0&limit=5>; rel="first",
<https://www.myhost.com/sample/api/v1/books?offset=5&limit=5>; rel="prev",
```

8. Versiona la tua API

Rendere la versione dell'API obbligatoria e non rilasciare mai una API senza una versione. Il modo più comune è quello di utilizzare una parte dell'URL per la versione oppure passarla come parametro (obbligatorio). E' consigliabile utilizzare un semplice numero ordinale per evitare la notazione col punto, come 2 . 5.

```
/sample/api/v1/books
/sample/api/books?v=1
```

9. Gestisci gli errori usando i codici di stato HTTP

Lo standard HTTP fornisce più di 70 codici di stato per descrivere i valori di ritorno. Non abbiamo bisogno di tutti questi, ma diciamo che almeno 10 ci tornano utili.

200 – OK – Tutto bene
201 – OK – E' stata creata una nuova risorsa
204 – OK – La risorsa è stata cancellata con successo
304 – Not modified – I dati non sono cambiati. Il cliente può utilizzare i dati nella cache
400 – Bad Request – Richiesta non valida. L'errore esatto dovrebbe essere spiegato nel payload dell' errore (di cui ne parleremo a breve). Per esempio. "Il JSON non è valido"
401 – Unauthorized – La richiesta richiede una autenticazione dell'utente
403 – Forbidden – Il server ha capito la richiesta, ma in base ai diritti del richiedente l'accesso non è consentito.
404 – Not Found – Non vi è alcuna risorsa dietro l'URI richiesto.
422 – Unprocessable Entity – deve essere usato se il server non può elaborare il enitity, ad esempio se un'immagine non può essere formattata o campi obbligatori sono mancanti nel payload.
500 – Internal Server Error – gli sviluppatori di API dovrebbero evitare questo errore. Se si verifica un errore globale dell'applicazione, lo stacktrace deve esere loggato e non inviato nella risposta all'utente.

Usa il payload dell'errore

Tutte le eccezioni dovrebbero essere mappate in un payload dell'errore. Ecco un esempio di come dovrebbe essere un payload in un JSON:

```
{
  "errors": [
   {
    "userMessage": "Sorry, the requested resource does not exist",
    "internalMessage": "No book found in the database",
    "code": 34,
    "more info": "http://www.myhost.com/blog/api/v1/errors/34"
   }
  ]
}
```

mer 10 febbraio 2021 - Lo Sviluppatore anno VI

10. Consenti l'override dei metodi HTTP

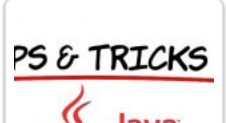
Alcuni proxy supportano solo i metodi POST e GET. Per sostenere una API RESTful con queste limitazioni, l'API ha bisogno di un modo per fare l'ovveride dei metodi HTTP.

Settare l'header HTTP non standard *X-HTTP-Method-override* nella request per fare l'ovveride.

Riferimenti Utili

- Representational State Transfer
- List of HTTP header fields
- Codici di stato HTTP
- RESTful Web Services – La Guida
- Representational State Transfer (REST)

Potrebbe interessarti anche:



Tags best practices restful

‹ Articolo Precedente
Java: Le espressioni regolari

Articolo Successivo ›
VIM: l'editor di testo che tutti i programmatori dovrebbero conoscere

Lascia un commento

Comment *

Name *

Email Address *

Website

Do il mio consenso affinché un cookie salvi i miei dati (nome, email, sito

mer 10 febbraio 2021 – **Lo Sviluppatore** anno VI

commento.

Pubblica il commento

SEGUI LO SVILUPPATORE SU...

CITAZIONI

LA VIGNETTA

mer 10 febbraio 2021 – Lo Sviluppatore anno VI

LE RECENSIONI IMPROBABILI