Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Progettazione REST

2022

Lorenzo Sommaruga

Contenuti ed Obiettivi

Contenuti

- Proprietà dei metodi HTTP
- Progettazione di API per sviluppo di un web services in REST
- Pattern REST

Obiettivi

- (BI-3) Comprendere le proprietà Safe, Idempotent, Cacheable dei metodi HTTP
- (BI-3) Comprendere buone pratiche per lo sviluppo di API REST
- (BI-3) Comprendere i principali pattern di progettazione REST

Principali metodi HTTP

- Ref. Specifica standard: https://tools.ietf.org/html/rfc7231#section-4

Method	Description	Sec.
GET	Transfer a current representation of the target resource.	4.3.1
HEAD	Same as GET, but only transfer the status line and header section.	4.3.2
POST	Perform resource-specific processing on the request payload.	4.3.3
PUT	Replace all current representations of the target resource with the request payload.	4.3.4
DELETE	Remove all current representations of the target resource.	4.3.5
CONNECT	Establish a tunnel to the server identified by the target resource.	4.3.6
OPTIONS	Describe the communication options for the target resource.	4.3.7
TRACE	Perform a message loop-back test along the path to the target resource.	4.3.8

REST - HTTP access pattern

- **CRUD via HTTP**
- **Idea base è di una avere una Resource, con un URI, una Representation, un insieme standard codificato di Operations**
- **Principali OPERAZIONI e metodi corrispondenti:**

Operazione	Descrizione	Metodo HTTP
Create	Crea una nuova risorsa	POST
Read	Trasferisce la risorsa senza effetti collaterali	GET
Update	Modifica il valore di una risorsa	PUT
Delete	Elimina risorsa	DELETE

Proprietà dei metodi HTTP

- I metodi HTTP hanno delle proprietà interessanti per il programmatore:
  - Safe
  - Idempotente
  - Cacheable

Def. Proprietà Safe

- **Def. Metodi Safe**
- *"safe" se semantica read-only*
- non alterano mai le risorse

- Es. GET è safe

- Il client non richiede né si aspetta nessun cambiamento sul server applicando il metodo alla risorsa target

Def. Proprietà Idempotente

- **Def. Metodi Idempotent**

"idempotent" se gli effetti, previsti sul server, di successive richieste identiche con quel metodo sono gli stessi di una singola richiesta

- GET, PUT, DELETE, sono *idempotent*
- Le richieste *idempotent* possono essere ripetute automaticamente senza effeti collaterali, e.g. in caso di problemi di comunicazione (non arriva risposta al client).
- E.g. Ripetizioni di:

  GET /book

  PUT /order/123

  DELETE /order/4

Def. Proprietà Cacheable

- **Def. Metodi Cacheable**

"cacheable" indica che le risposte possono essere memorizzate per ri-usi futuri

- In generale i metodi *"safe"* che non dipendono da risposte sono *cacheable*

- GET, HEAD, ~~(POST impl. dip.)~~, sono *cacheable*

Proprietà dei metodi HTTP

- Ref. Specifica standard: https://tools.ietf.org/html/rfc7231#section-8.1

8.1.3. Registrations

The "Hypertext Transfer Protocol (HTTP) Method Registry" populated with the registrations below:

Method	Safe	Idempotent	Reference
CONNECT	no	no	Section 4.3.6
DELETE	no	yes	Section 4.3.5
GET	yes	yes	Section 4.3.1
HEAD	yes	yes	Section 4.3.2
OPTIONS	yes	yes	Section 4.3.7
POST	no	no	Section 4.3.3
PUT	no	yes	Section 4.3.4
TRACE	yes	yes	Section 4.3.8

Progettazione REST

- **Alcune linee guida per una buona progettazione di API REST**
  - Nomi delle risorse e URI
  - Documentazione delle API
  - Versioni
  - HTTP CRUD pattern
  - Uso dei codici di risposta HTTP
  - Parametri aggiuntivi in richieste
  - Alcuni Patterns e Anti-patterns

Terminologia - premessa

- **Risorsa** specifica: oggetto o rappresentazione di entità significative nel dominio applicativo

  E.g. un Libro, un Ordine, un BlogPost

- **Collezione**: insieme di risorse omogenee

  E.g. Catalogo è una collezione di risorse Libro

- **URI:** è il path della risorsa con il quale si possono eseguire operazioni

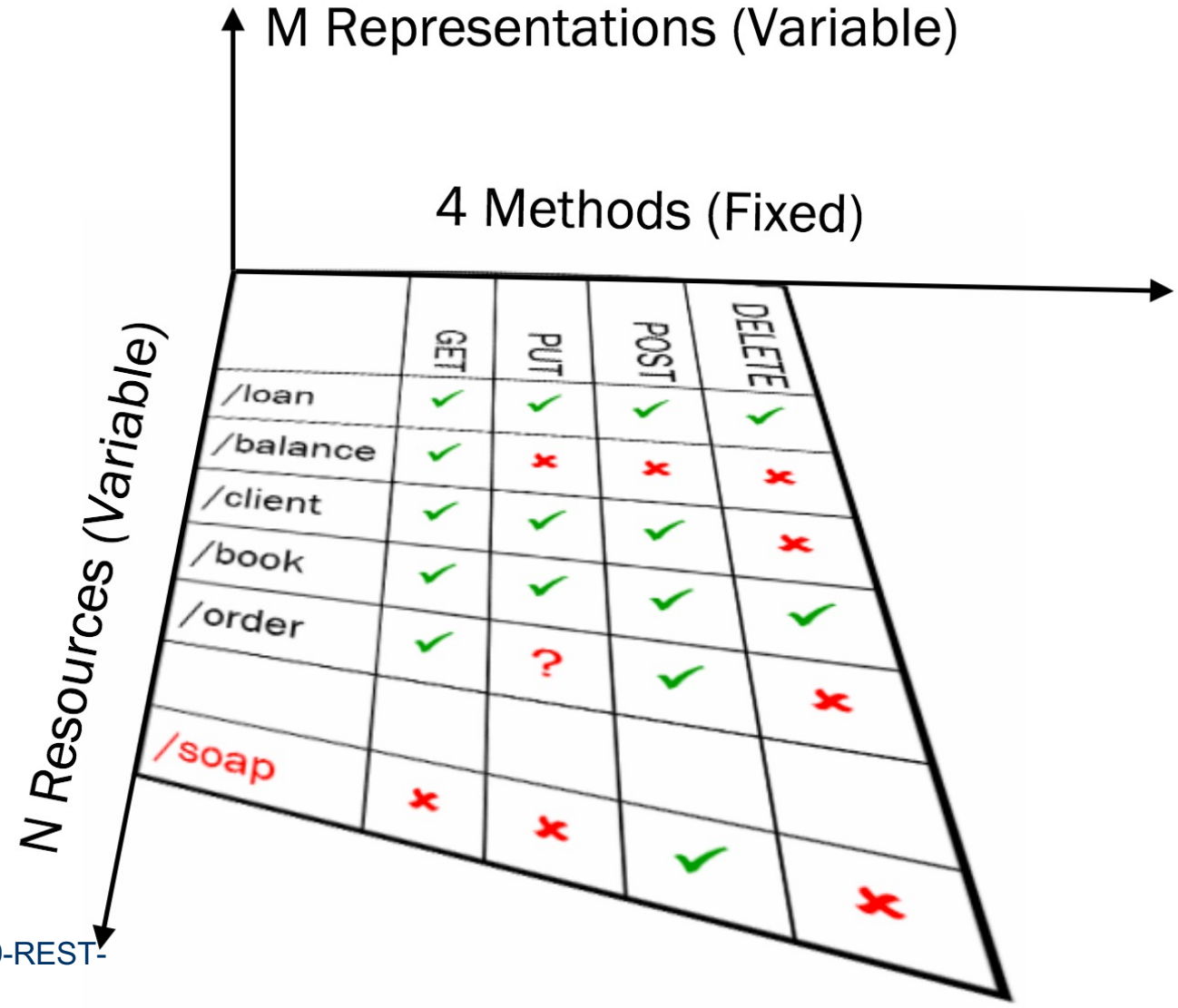
Passi Progettazione REST

1. **Identificare le risorse** da esporre per il servizio web (e.g., book catalog, purchase order, polls and votes, blogpost)
2. **Modellare le relazioni** (e.g., containment/sub-part, reference, state transitions) tra le risorse con hyperlinks (HATEOAS)
3. **Definire URIs** corretti per l'indirizzamento delle risorse
4. **Comprendere il significato** di esecuzione dei metodi (GET, POST, PUT, DELETE) su ogni risorsa, e se sia o meno permesso
5. **Progettare** e **documentare** la rappresentazione delle risorse
6. **Implementare e fare deploy** su (Web) server
7. **Testare** in un Web browser

Ref. Libro: SOA with REST Principles, Patterns …

Esempio Progettazione REST

- Utile la definizione di una tabella



Fonte C. Pautasso 2009
http://www.jopera.org/files/SOA2009-REST-Patterns.pdf

Patterns

- Molti diversi patterns

1) Statelessness
2) Content Negotiation
3) URI Templates
4) Design for Intent
5) Versioning
6) Authorization
7) Bulk Operations
8) Pagination

1. Uniform Contract
2. Entity Endpoint
3. Entity Linking
4. Content Negotiation
5. Distributed Response Caching
6. Endpoint Redirection
7. Idempotent Capability
8. Message-based State Deferral
9. Message-based Logic Deferral
10. Consumer-Processed Composition

Refs.: https://www.quora.com/What-are-some-good-design-patterns-for-RESTful-Services
https://www.snyxius.com/restful-api-design-patterns/

Patterns: Creazione di risorse

- **Factory resource** URI
  - Factory resource rappresentano il "type" di risorsa
  - Hanno un URI statico, rappresentato dalla forma **plurale** del tipo di risorsa
  - E.g. http://a.b.c/books
http://a.b.c/users
http://a.b.c/companies
- **Istanza di risorsa**
  - rappresentano una istanza del tipo di risorsa
  - Hanno un ID univoco nell'URI per localizzare la risorsa
  - E.g. http://a.b.c/books/123
http://a.b.c/users/4
- **Risorsa dipendente**
  - rappresentano sotto parti, prefisso di risorsa parent
  - Hanno un ciclo di vita dipendente dalla risorsa parent
  - Delete di risorsa -> delete di risorse dipendenti
  - E.g. http://a.b.c/books/123/tableofcontent
http://a.b.c/users/shoppingcart

Patterns: Creazione di risorse - PUT e POST

- **PUT**
  - Crea o aggiorna la *risorsa* specifica identificata dall'URI
    - E.g. PUT http://a.b.c/books/123
  - Se esiste già il server aggiorna la risorsa

    altrimenti la crea associandola all'URI specificato
    - Se il server non può accettare che il client decida URI di nuova risorsa -> restituisce errore (e.g. 405)
  - Generalmente l'URI di PUT è una *risorsa* specifica, non è di tipo *collezione*
- **POST**
  - crea una nuova istanza di risorsa di tipo *collezione*
    - E.g. POST http://a.b.c/books
  - Il tipo è identificato dall'URI della richiesta
    - URI rapresenta il parent della risorsa -> tipo *collezione*
  - In caso di successo la risposta contiene URI di risorsa creata
  - Generalmente l'URI di POST è di tipo *collezione*
    - -> il client vuole aggiungere una nuova risorsa nella *collection* specificata

Patterns: Nomi di URI per risorse

- Usare nomi plurali (risorse)
  e non verbi (azioni in metodi HTTP)
- URI brevi
- Passaggio di parametri “positional” non key=value&prop=val
- Uso di suffissi per specificare content type ≠
- NON cambiare URI, usare redirection se necessario

Esempi di Nomi di URI per risorse

- Usare il plurale e aggiungere ID per accedere ad una istanza di risorsa
- Esempi
- GET `/companies` lista di tutte le companies
- GET `/companies/3` dettagli della company 3
- DELETE `/companies/3` delete della company 3
- GET `/companies/3/employees` lista di tutti gli employees della company 3
- GET `/companies/3/employees/45` dettagli dell'employee 45, appartenente alla company 3
- DELETE `/companies/3/employees/45` delete dell'employee 45, della company 3

Patterns: Rappresentazione risorse Content Negotiation

- Utile per esporre rappresentazioni multiple della risorsa. 3 modi:

1. content negotiation : Accept e Content-Type

2. Uso di un URI distinto per ogni rappresentazione
   - E.g. http://a.b.c/books_json
     http://a.b.c/books_xml

3. Approccio misto: richieste con content negotiation a risorse che vengono redirette a URI specifico

- Vantaggi: Loose Coupling, > interoperabilità
- Una buona applicazione dovrebbe permettere di poter cambiare il formato delle risposte per adattarsi alle esigenze dei client

Patterns: Content Negotiation Esempio

- La negoziazione del formato non necessita altre richieste
- Il client elenca i formati accettati (MIME)
- Il server sceglie il formato più appropriato per la risposta
  - Può ritornare codice 406 se non dispone di nessun formato

- Esempio

Richiesta

```
GET /books/123
Accept: text/html, application/xml, application/json
```

Risposta

```
200 OK
Content-Type: application/xml
```

Codici di risposta 2xx

- Dopo una richiesta al server il client dovrebbe saper interpretare il feedback
- utilizzare i codici HTTP in modo standard

- **Principali codici**

- 2xx (codici di SUCCESSO)
  - sono codici utilizzati per confermare che la richiesta è stata ricevuta e processata con successo dal server
    - **200 Ok**: risposta standard per successo di operazioni GET, PUT or POST (per azioni e non creazione risorsa); ritorna una rappresentazione-risorsa
    - **201 Created**: comunica che la nuova istanza è stata creata con successo (eg. nuova risorsa con POST)
    - **202 Accepted**: l'azione avrà probabile successo ma non è stata ancora eseguita
    - **204 No Content**: la richiesta è stata correttamente processata, ma non ha ritornato nessun contenuto. E.g. DELETE ottimo esempio

Esempio Codice 200 OK

Ref sec. 15.3.1. 200 OK

The _200 (OK)_ status code indicates that the request has succeeded. The content sent in a 200 response depends on the request method. For the methods defined by this specification, the intended meaning of the content can be summarized as:

Fielding, et al. Expires 14 March 2022 [Page 150] Internet-Draft HTTP Semantics September 202

request method	response content is a representation of
GET	the target resource
HEAD	the target resource, like GET, but without transferring the representation data
POST	the status of, or results obtained from, the action
PUT, DELETE	the status of the action
OPTIONS	communication options for the target resource
TRACE.	the request message as received by the server returning the trace

Codici di risposta 3xx

- 3xx (codici di Redirection)
  - **304 Not Modified**: il client ha già in cache la risposta e non è necessario reinviarla
  - **302** Moved Temporarily: effettua una redirection
    cambiato in 302 Found e distinto in 303 See Other -> GET e 307 Temporary Redirect -> preserva metodo

- Esempio

Richiesta:

```
GET /index.html HTTP/1.1
Host: www.example.com
```

Risposta:

```
HTTP/1.1 302 Found Location:
http://www.iana.org/domains/example/
```

Codici di risposta 4xx

- 4xx (codici di errore client)
  - concerne errori correttamente gestiti dal server e causati da un possibile input non corretto della richiesta

    - **400 Bad Request**: la richiesta non è stata processata perché il server non riconosce la richiesta: e.g. uri o nome parametro errato
    - **401 Unauthorized**: il client non ha i privilegi per accedere alla risorsa: e.g. le API richiedono l'autenticazione o un token
    - **403 Forbidden**: la richiesta è valida e il client è correttamente autenticato, ma non ha i privilegi per poter utilizzare l' API specificata
    - **404 Not Found**: la risorsa non esiste o è temporaneamente non accessibile
    - **410 Gone**: la risorsa non è disponibile ed è stata spostata

Codici di risposta 5xx

- 5xx (codici di errore server)
  - rappresentano situazioni non correttamente gestite come eccezioni lato server oppure servizi momentaneamente non disponibili

    - **500 Internal Server Error**: indica che la richiesta è valida, ma il server ha riscontrato un errore non gestito (ad es. in java una eccezione non correttamente catturata).
    - **503 Service Unavailable**: indica che il server è offline e non può quindi processare richieste.

Registro dei codici di Stato HTTP

- Hypertext Transfer Protocol (HTTP) Status Code Registry (Last Updated 2022-06-08)
- Ref.:

https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml

Value	Description	Reference
100	Continue	[RFC7231, Section 6.2.1]
101	Switching Protocols	[RFC7231, Section 6.2.2]
102	Processing	[RFC2518]
103	Early Hints	[RFC8297]
104-199	Unassigned	
200	OK	[RFC7231, Section 6.3.1]
201	Created	[RFC7231, Section 6.3.2]
202	Accepted	[RFC7231, Section 6.3.3]
203	Non-Authoritative Information	[RFC7231, Section 6.3.4]
204	No Content	[RFC7231, Section 6.3.5]
205	Reset Content	[RFC7231, Section 6.3.6]
206	Partial Content	[RFC7233, Section 4.1]
207	Multi-Status	[RFC4918]
208	Already Reported	[RFC5842]
209-225	Unassigned	

Esempi uso codici stato HTTP

- Ref. Specification: https://tools.ietf.org/html/rfc7231#section-4
- Se Request method è ricevuto e non riconosciuto o non implementato dal server -> il server SHOULD respond con **501** (Not Implemented)
- Se Request method è ricevuto e riconosciuto ma non permesso dal server per la risorsa target -> il server SHOULD respond con **405** (Method Not Allowed)
- Esempio DELETE: 200 (OK), 204 (No Content), 404 (Not Found)

  Richiesta: `DELETE /item/12`

  Risposte:

  - Prima volta: se esiste risorsa risponde con 200 (OK) o 204 (No Content)
  - Chiamate successive: 404 (Not Found)

  È idempotente, quindi lo stato sul server è lo stesso dopo ogni richiesta DELETE, ma … la risposta (status) è diversa.

  Attenzione: se DELETE decrementa un contatore -> NON idempotente

Patterns: Versioni

- Permette di gestire modifiche alle API
- Aggiungere un numero di versione identificativo alla API

1. **Soluzione esplicita**
   - Nel path delle API
   - E.g.
     - `/v1/books/123`
     - `/v2/books/123`
   - Due URI per stessa risorsa?
2. **Soluzione implicita**
   - Uso di accept header per distinguere versioni

Anti patterns

- **Tutto in GET**
  - Non *idempotent* e non *safe*, e.g:

    GET /users?action=deleteUser&id=123

- **Tutto in POST**
  - Non *idempotent* e non *safe*, non *cacheable*

Riferimento

- **Metodi HTTP**
  - HTTP request methods https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
  - Intro a 7 metodi HTTP e Testing https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them
- **Idempotenza**
  - Idempotent Capability http://soapatterns.org/design_patterns/idempotent_capability
  - Discussione: What Is Idempotent in REST? https://www.infoq.com/news/2013/04/idempotent
- **Patterns**
- RESTful API Design Patterns: Everything You Need to Know https://www.snyxius.com/restful-api-design-patterns/

- **Libri**
- SOA with REST Principles, Patterns & Constraints for Building Enterprise Solutions with REST (paperback) (The Prentice Hall Service Technology Series from Thomas Erl) 1st Edition https://www.amazon.com/Principles-Constraints-Enterprise-Solutions-Technology/dp/0134767446

- Testing web service: https://assertible.com/