Raccomandazioni di implementazione

Versione 1.0 del 27/04/2021

AGID | via Liszt, 21 – 00144 Roma | protocollo@pec.agid.gov.it | www.agid.gov.it | tel. 06 85264.1

Versione	Data	Tipologia modifica
1	27/04/2021	Prima emissione

Sommario

Introduzione

Il presente documento operativo riporta le indicazioni che gli erogatori considerano nell'implementazione delle API al fine di favorire l'interoperabilità con i fruitori.

Le raccomandazioni sono applicate dagli erogatori in funzione alle specifiche esigenze applicative ed in relazione alla natura dei fruitori.

Capitolo 1

Ambito di applicazione

Il presente Documento operativo è redatto quale documento operativo relativo alla Linee Guida sull'interoperabilità tecnica.

1.1 Soggetti destinatari

Il Documento Operativo è destinato ai soggetti di cui all'articolo 2, comma 2 del CAD, così come indicato dall'articolo 75 dello stesso. I destinatari la attuano nella realizzazione dei propri sistemi informatici che fruiscono o erogano dati e/o servizi digitali di/ad altri soggetti.

Per i servizi implementati dagli erogatori prima dell'emanazione del Documento Operativo, al fine di assicurare la convergenza al ModI, si richiede di:

- assicurare per i nuovi fruitori l'applicazione di modalità di fruizione conformi al Documento Operativo;
- prevedere, a valle di una valutazione di impatto che includa l'effetto sui fruitori, la dismissione delle modalità difformi al Documento Operativo.

Il Documento Operativo è rivolto ai soggetti privati che devono interoperare con la Pubblica Amministrazione per erogare o fruire di dati e servizi tramite sistemi informatici.

Capitolo 2

Riferimenti e sigle

2.1 Note di lettura del documento

Conformemente alle norme ISO/IEC Directives, Part 3 per la stesura dei documenti tecnici la presente linea guida utilizzerà le parole chiave «DEVE», «DEVONO», «NON DEVE», «NON DEVONO», «E" RICHIESTO», «DOVREBBE», «NON DOVREBBE», «RACCOMANDATO», «NON RACCOMANDATO» «PUO"» e «OPZIONALE», la cui interpretazione è descritta di seguito.

- **DEVE** o **DEVONO**, indicano un requisito obbligatorio per rispettare la linea guida;
- **NON DEVE** o **NON DEVONO**, indicano un assoluto divieto delle specifiche;
- **DOVREBBE** o **RACCOMANDATO** o **NON DOVREBBE** o **NON RACCOMANDATO**, indicano che le implicazioni devono essere comprese e attentamente pesate prima di scegliere approcci alternativi;
- **PUÓ** o **POSSONO** o l'aggettivo **OPZIONALE**, indica che il lettore può scegliere di applicare o meno senza alcun tipo di implicazione la specifica.

2.2 Standard di riferimento

Sono riportati di seguito gli standard tecnici indispensabili per l'applicazione del presente documento.

[IEEE 754]	Standard per il calcolo in virgola mobile
[ISO 639]	Standard che elenca i codici brevi per l'individuazione delle maggiori lingue
[ISO 3166]	Standard che definisce i codici per i nomi dei Paesi
[ISO 4217]	Standard internazionale per identificare le valute
[ISO 8601]	Standard per una rappresentazione numerica di date e orari
[ISO 60559]	Standard per la aritmetica a virgola mobile
[RFC3339]	Date and Time on the Internet
[RFC6838]	Media Type Specifications and Registration Procedures
[RFC8259]	JavaScript Object Notation (JSON) Data Interchange

[RFC7231]	Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content
[RFC7232]	Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests
[RFC7234]	Hypertext Transfer Protocol (HTTP/1.1): Caching
[RFC7807]	REST Problem Details
[RFC8288]	Web Linking

2.3 Termini e definizioni

Di seguito si riportano gli ACRONIMI che verranno utilizzati nella presente Linee Guida:

[AgID]	Agenzia per l'Italia Digitale
[API]	Application programming interface
[CAD]	Codice Amministrazione Digitale, D.Lgs 7 marzo 2005, n. 82
[HTTP]	Hypertext Transfer Protocol
[IDL]	Interface Description Language
[JSON]	JavaScript Object Notation
[ModI]	Modello di Interoperabilità
[MTOM]	Message Transmission Optimization Mechanism
[OPENAPI]	Specifica per gestire servizi web RESTful
[REST]	Representational State Transfer
[SOAP]	Simple Object Access Protocol
[SWA]	SOAP with Attachments
[URI]	Uniform Resource Identifier
[UTC]	Universal Time Coordinated
[UTF-8]	Unicode Transformation Format, 8 bit
[XOP]	XML-binary Optimized Packaging
[W3C]	World Wide Web Consortium
[WSDL]	Web Services Description Language
[WS-I]	Web service Basic Profile versione 2

Capitolo 3

Raccomandazioni tecniche generali

In quanto segue sono riportate le raccomandazione tecniche che gli erogatori considerano nell'implementazione delle proprie API.

3.1 Raccomandazioni globali

3.1.1 [RAC_GEN_001] Descrizione delle API

Le API DEVONO essere rappresentate mediante un Interface Description Language standard (IDL). Nello specifico:

- per REST, OpenAPI 3.0 e successive;
- per SOAP, WSDL 1.1 e successive.

3.1.2 [RAC_GEN_002] Endpoint delle API

Il numero di versione NON DEVE essere presente all'interno del nome della API.

Si DOVREBBE indicare il numero di versione e la tecnologia nell'endpoint delle API.

Esempio:

```
https://<dominioOrganizzativo>/[rest|soap]/<DominioApplicativo>/v<major>[.<minor>[.<patch>]]/<Nome API>
```

dove:

- <dominioOrganizzativo> indica l'organizzazione che espone il servizio;
- [rest|soap] indica la tecnologia della API;
- <DominioApplicativo> indica il settore all'interno dell'organizzazione;
- v<major>[.<minor>[.<patch>]] indica il numero di versione in coerenza con Semantic Versioning 2.0.0 1;
- <NomeAPI> è il nome della specifica API.

Il documento di specifica dell'API DEVE indicare la versione, includendo <major>.<minor>.<patch>.

Listato 4.1 Un esempio di versioning in formato OAS3

```
openapi: 3.0.3
info:
  version: 1.3.4
...
```

3.1.3 [RAC_GEN_003] Codifica di default

Si DOVREBBE utilizzare UTF-8 come codifica di default per i dati.

3.1.4 [RAC_GEN_004] Non passare credenziali o dati riservati nell'URL

Eventuali dati riservati o credenziali e token di autenticazione NON DEVONO essere passati nei query parameters o comunque nell'URL.

3.2 Raccomandazioni sul formato dei dati

3.2.1 [RAC_GEN_FORMAT_001] Utilizzare Content-Type semanticamente coerenti

Quando si ritornano dati binari, immagini o documenti (eg. pdf, png, …) si DEVONO utilizzare i rispettivi Content-Type. Nel protocollo HTTP, l'HTTP header Content-Type indica il media-type di una risorsa.

3.2.2 [RAC_GEN_FORMAT_002] Evitare Content-Type personalizzati

Si DOVREBBE evitare l'uso di media-type personalizzati come da RFC 6838#section-3.4 (eg. application/x.custom.name+xml, application/x.custom.name+json) ed utilizzare nomi standard come:

- application/xml
- application/soap+xml
- application/json
- application/problem+json
- application/jose+json

L'elenco dei media-type è reperibile sul sito IANA[1].

3.2.3 [RAC_GEN_FORMAT_003] Formati standard per Data ed Ora

Le date DEVONO essere conformi:

- alla sintassi «full-date» indicata in RFC 3339, ad esempio 2015-05-28 se si indica una data;
- alla sintassi «date-time» indicata in RFC 3339, ad esempio 2015-05-28T14:07:17Z o nel formato UNIX Timestamp definito in «The Open Group Base Specifications Issue 7, Rationale: Base Definitions, section A.4 General Concepts» se si indica un momento esatto nel tempo.

RFC 3339 permette di indicare una timezone prefissando la data con la distanza da UTC:

- 2015-05-28T14:07:17+01:00
- 2015-05-28T14:07:17-05:00

Quando la data è specificata in UTC occorre utilizzare sempre il suffisso Z (Zulu time zone):

- 2015-05-28T14:07:17Z

Ove non dettagliato nelle specifiche, le date negli header HTTP DEVONO essere conformi:

- al formato UNIX Timestamp;
- alla sintassi HTTP-date definito in RFC 7231, eg. «Sun, 06 Nov 1994 08:49:37 GMT».

3.2.4 [RAC_GEN_FORMAT_004] Tempi di durata e intervalli

I tempi di durata e gli intervalli DEVONO essere espresse in secondi o usare lo standard ISO 8601.

Di seguito alcuni esempi di durata in formato ISO 8601.

I tempi di durata sono prefissati da «P»; giorni e ore sono separati da «T».

[1] Cf. https://www.iana.org/assignments/media-types/media-types.xhtml

Esempi:

- P1Y2M3D - 1 anno, 2 mesi e 3 giorni
- PT1H4M5S - 1 ora, 4 minuti e 5 secondi
- P1M - 1 mese
- PT1M - 1 minuto
- P1Y2M10DT2H30M - 1 anno, 2 mesi, 10 giorni 2 ore e 30 minuti

3.2.5 [RAC_GEN_FORMAT_005] Lingue e monete

Si DEVONO utilizzare per le codifiche standard indicate nelle Linee Guida per la Valorizzazione del Patrimonio Informativo Nazionale[2], inclusi:

- ISO 3166-1-alpha2 country (due lettere)
- ISO 639-1 language code
- ISO 639-1 per le varianti dei linguaggi.
- ISO 4217 alpha-3 currency codes

Per le valute è possibile basarsi sullo schema Money - ripreso dal lavoro di standardizzazione del Berlin Group sotto l'egida dello European Standards e contenente i campi:

- amount (string)
- currency [ISO-4217]

Esempio:

```
{
        "tax_id": "imu-e472",
        "value": {
                "amount": "100.23",
                "currency": "EUR"
        }
}
```

```
<payment>
```

[2] Cf. https://docs.italia.it/italia/daf/lg-patrimonio-pubblico/it/bozza/index.html

```
        <taxId>imu-e472</taxId>
        <value>
                <currency>EUR</currency>
                <amount>100.23</amount>
        </value>
</payment>
```

3.3 Raccomandazioni sulla progettazione e naming

3.3.1 [RAC_GEN_NAME_001] Utilizzare i nomi delle proprietà secondo nomenclature standard

Le proprietà DEVONO utilizzare, ove possibile, la nomenclatura indicata nelle Linee Guida per la valorizzazione del Patrimonio[3] informativo pubblico e le relative ontologie[4].

3.3.2 [RAC_GEN_NAME_002] Nomenclatura delle proprietà

Le proprietà DOVREBBERO avere una nomenclatura consistente.

Scegliere uno dei due stili di seguito e modificarlo in ASCII:

- snake_case
- camelCase

Non usare contemporaneamente snake_case e camelCase nella stessa API.

Ad esempio:

SI

```
{
  "givenName": "Mario",
  "familyName": "Rossi"
}
```

SI

[3] Cf. https://docs.italia.it/italia/daf/lg-patrimonio-pubblico/it/bozza/index.html
[4] Cf. https://github.com/italia/daf-ontologie-vocabolari-controllati

```
{
  "given_name": "Mario",
  "family_name": "Rossi"
}
```

NO

```
{
  "given_name": "Mario",
  "familyName": "Rossi"
}
```

SI

```
<givenName>Mario</givenName>
<familyName>Rossi</familyName>
```

SI

```
<given_name>Mario</given_name>
<family_name>Rossi</family_name>
```

NO

```
<given_name>Mario</given_name>
<familyName>Rossi</familyName>
```

3.3.3 [RAC_GEN_NAME_003] Descrittività dei nomi utilizzati

I nomi utilizzati per servizi, path, operation o schemi DEVONO essere auto-descrittivi e fornire quanta più informazione utile riguardo al comportamento implementato, evitando però le ridondanze.

Si deve inoltre evitare l'utilizzo di acronimi quando questi non siano universalmente riconosciuti anche al di fuori del dominio applicativo.Esempio in un'architettura orientata alle risorse:

In un servizio per la gestione delle istanze dei cittadini, il nome dell'attributo

gestioneIstanzeCittadinoAbilitatoBoolean

può essere semplificato in

cittadinoAbilitato

se il servizio è limitato alla gestione delle istanze e l'output del campo è desumibile dal contesto.

3.4 Raccomandazioni sul logging

3.4.1 [RAC_GEN_LOG_01] Informazioni di Logging

I log file DEVONO contenere:

- l'istante della comunicazione in formato UTC (RFC 3339) e con i separatori Z e T in maiuscolo. La specifica è fondamentale per l'interoperabilità dei sistemi di logging e auditing, evitando problemi di transizione all'ora legale e la complessità nella gestione di fusi orari nell'ottica dell'interoperabilità con altre PA europee;
- l'URI che identifica l'erogatore e l'operazione richiesta;
- la tipologia di chiamata (ad esempio, metodo HTTP per i protocolli basati su HTTP);
- esito della chiamata (ad es., stato della response HTTP se disponibile, SOAP fault nel caso di web service SOAP);
- ove applicabile, l'Indirizzo IP del client;
- ove applicabile, identificativo del consumatore o altro soggetto operante la richiesta comunicato dal fruitore. È cura del fruitore procedere alla codifica e all'anonimizzazione, ove necessario;
- ove applicabile, un identificativo univoco della richiesta, utile a eventuali correlazioni.

3.5 Raccomandazioni sulla robustezza

Ai fini di garantire la responsività di una API è necessario impedire a singoli fruitori di esaurire la capacità di calcolo e di banda dell'erogatore. La tecnica comunemente utilizzata in questi casi è il rate limiting (anche noto come throttling). Il rate limit fornisce ad uno specifico

fruitore un numero massimo di richieste soddisfacibili all'interno di uno specifico arco temporale (es. 1000 richieste al minuto). Un numero di richieste che superi il limite imposto provoca il rifiuto di ulteriori richieste da parte di uno specifico fruitore per un intervallo di tempo predeterminato.

Sulle politiche riguardanti il numero massimo di richieste e la relativa finestra temporale, e quelle riguardanti il tempo di attesa per nuove richieste (che può essere incrementato in caso di richieste reiterate, es. con una politica di aumento esponenziale) si lascia libertà agli implementatoti previa un'analisi di carico massimo sopportabile dall'erogatore.

3.5.1 [RAC_ROBUSTEZZA_001] Segnalare raggiunti limiti di utilizzo

Gli erogatori di interfacce di servizio REST DEVONO segnalare eventuali limiti raggiunti con HTTP status 429 Too Many Requests.

Le API restituiscono in ogni risposta i valori globali di throttling tramite i seguenti header[5]:

- X-RateLimit-Limit: limite massimo di richieste per un endpoint;
- X-RateLimit-Remaining: numero di richieste rimanenti fino al prossimo reset;
- X-RateLimit-Reset: numero di secondi che mancano al prossimo reset.

In caso di superamento delle quote, le API DEVONO restituire anche l'header:

- HTTP header Retry-After : numero minimo di secondi dopo cui il client è invitato a riprovare[6].

Nel caso di SOAP non esistono regole guida standard per la gestione del rate limit e del throttling. Si POSSONO utilizzare gli stessi header e status code HTTP visti nel caso REST.

I fruitori DEVONO:

- rispettare gli header di throttling;
- rispettare l'header X-RateLimit-Reset quando restituisce il numero di secondi che mancano al prossimo reset, ed eventualmente gestire l'indicazione in timestamp unix;

[5] È in corso il processo di standardizzazione dell'utilizzo degli header indicati https://datatracker.ietf.org/doc/draft-ietf-httpapi-ratelimit-headers/

[6] Cf. RFC 7231 prevede che l'header HTTP header Retry-After possa essere utilizzato sia in forma di data che di secondi

- rispettare l'header HTTP header Retry-After sia nella variante che espone il numero di secondi dopo cui riprovare, sia nella variante che espone la data in cui riprovare.

3.5.2 [RAC_ROBUSTEZZA_002] Segnalare il sovraccarico del sistema o l'indisponibilità del servizio

Gli erogatori DEVONO definire ed esporre un piano di continuità operativa segnalando il sovraccarico del sistema o l'indisponibilità del servizio con HTTP status 503 Service Unavailable Service Unavailable.

In caso di sovraccarico o indisponibilità, l'erogatore DEVE ritornare anche:

- HTTP header Retry-After con il numero minimo di secondi dopo cui il client è invitato a riprovare.

I fruitori DEVONO:

- rispettare HTTP header Retry-After sia nella variante che espone il numero di secondi dopo cui riprovare, sia nella variante che espone la data in cui riprovare.

Per REST si DEVE prevedere nella descrizione delle API l'indicazione degli header relativi al rate limiting. L'utilizzo degli header HTTP in SOAP è fuori dagli obiettivi di WSDL come Interface Definition Language.

Esempio di specifica API REST per applicazione del throttling e segnalazione del sovraccarico o dell'indisponibilità:

```
openapi: 3.0.1
info:
   title: RESTrobustezza
   license:
      name: Apache 2.0 License
      url: http://www.apache.org/licenses/LICENSE-2.0.html
   version: "1.0"
paths:
   /resources/{id_resource}/M:
      post:
         description: M
         operationId: PushMessage_1
         parameters:
         - name: id_resource
           in: path
           required: true
           schema:
            type: integer
            format: int32
```

```
            requestBody:
               content:
                  application/json:
                    schema:
                       $ref: '#/components/schemas/MType'
            responses:
               200:
                  description: Esecuzione di M avvenuta con successo
                  headers: &ratelimit_headers
                     X-RateLimit-Limit:
                        $ref: '#/components/headers/X-RateLimit-Limit'
                     X-RateLimit-Remaining:
                        $ref: '#/components/headers/X-RateLimit-Remaining'
                     X-RateLimit-Reset:
                        $ref: '#/components/headers/X-RateLimit-Reset'
                  content:
                     application/json:
                        schema:
                           $ref: '#/components/schemas/MResponseType'
               404:
                  description: Identificativo non trovato
                  headers:
                     <<: *ratelimit_headers
                  content:
                     application/json:
                        schema:
                           $ref: '#/components/schemas/ErrorMessage'
               400:
                  description: Richiesta malformata
                  headers:
                     <<: *ratelimit_headers
                  content:
                     application/json:
                        schema:
                           $ref: '#/components/schemas/ErrorMessage'
               429:
                  description: Limite di richieste raggiunto
                  headers:
                     Retry-After:
                        description: Limite massimo richieste
                        schema:
                           type: string
                  content:
                     application/json:
                        schema:
                           $ref: '#/components/schemas/ErrorMessage'
               500:
                  description: Errore interno avvenuto
                  content:
                     application/json:
                        schema:
                           $ref: '#/components/schemas/ErrorMessage'
      components:
         headers:
            Retry-After:
               description: |-
```

```
                  Retry contacting the endpoint \*at least\* after seconds.
                  See https://tools.ietf.org/html/rfc7231#section-7.1.3
               schema:
                  format: int32
                  type: integer
            X-RateLimit-Limit:
               description: The number of allowed requests in the current period
               schema:
                  format: int32
                  type: integer
            X-RateLimit-Remaining:
               description: The number of remaining requests in the current period
               schema:
                  format: int32
                  type: integer
            X-RateLimit-Reset:
               description: The number of seconds left in the current period
               schema:
                  format: int32
                  type: integer
      schemas:
         MType:
            type: object
            properties:
               a:
                  $ref: '#/components/schemas/AComplexType'
               b:
                  type: string
         MResponseType:
            type: object
            properties:
               c:
                  type: string
         AComplexType:
            type: object
            properties:
               a1s:
                  type: array
               items:
                  type: integer
                  format: int32
               a2:
                  type: string
         ErrorMessage:
            type: object
            properties:
               detail:
                  description: |
                     A human readable explanation specific to this occurrence of the
                     problem.
                  type: string
               instance:
                  description: |
                    An absolute URI that identifies the specific occurrence of the
                    problem.
                    It may or may not yield further information if dereferenced.
                  format: uri
```

```
      type: string
    status:
      description: |
        The HTTP status code generated by the origin server for this
        occurrence
        of the problem.
      exclusiveMaximum: true
      format: int32
      maximum: 600
      minimum: 100
      type: integer
    title:
      description: |
        A short, summary of the problem type. Written in english and
        readable
        for engineers (usually not suited for non technical stakeholders
        and not localized);
      example: Service Unavailable
      type: string
    type:
      default: about:blank
      description: |
        An absolute URI that identifies the problem type. When
        dereferenced,
        it SHOULD provide human-readable documentation for the problem
        type
        (e.g., using HTML).
      format: uri
      type: string
```

Di seguito, un esempio di chiamata alle API, con risposta nel caso in cui i limiti non siano ancora stati raggiunti e nel caso in cui invece il fruitore debba attendere per presentare nuove richieste.

Endpoint

https://api.ente.example/rest/nome-api/v1/resources/1234/M

1. Request

```
POST /rest/nome-api/v1/resources/1234/M HTTP/1.1
Host: api.ente.example
Content-Type: application/json

{
"a": {
        "a1": [1, "...", 2],
        "a2": "RGFuJ3MgVG9vbHMgYXJlIGNvb2wh"
        },
"b": "Stringa di esempio"
```

```
}
```

2. Response 200 con rate limiting

```
HTTP/1.1 200 OK
X-RateLimit-Limit: 30
X-RateLimit-Remaining: 11
X-RateLimit-Reset: 44

{"c" :"risultato"}
```

2. Response 429 Too Many Requests

```
HTTP/1.1 429 Too Many Requests
Content-Type: application/problem+json
Retry-After: 60

{
"status": 429,
"title": "Hai superato la quota di richieste."
}
```

2. Response 503 Service Unavailable

```
HTTP/1.1 503 Service Unavailable
Content-Type: application/problem+json
Retry-After: 3600

{
"status": 503,
"title": "Servizio in manutenzione."
}
```

3.5.3 [RAC_ROBUSTEZZA_003] Uniformità di Indicatori ed Obiettivi di Servizio

Gli SLI pubblicati DEVONO:

- utilizzare unità di misure referenziate dal Sistema Internazionale (ad esempio, secondi o bytes);

- indicare nel nome identificativo l'eventuale periodo di aggregazione con i soli suffissi s (secondi), m (minuti), d (giorni) e y (anni), utilizzando al posto dei mesi il numero di giorni;
- includere la latenza aggiuntiva dovuta ad eventuali componenti infrastrutturali e di rete (ad esempio, proxy o gateway).

Gli SLO e gli SLA DOVREBBERO essere in relazione diretta con i valori associati (ad esempio, indicare il success rate anziché l'error rate), in modo che a valori più alti corrispondano risultati positivi.

Alcuni esempi di indicatori a cui è possibile associare degli obiettivi o degli accordi:

- dimensione massima di ogni richiesta accettata. Le richieste più grandi possono essere rifiutate
- latenza al 90º percentile. Utilizzata per calcolare la responsività
- percentuale di minuti negli ultimi 30 giorni in cui l'interfaccia di servizio è stata disponibile
- valori a 30 giorni del success rate, ovvero il numero di chiamate terminate con successo rispetto al numero totale di chiamate
- Application Performance inDEX[7], indice su scala percentuale di qualità del servizio misurato a 30 giorni
- tempo di risposta medio delle richieste totali (includendo le richieste rifiutate a causa del throttling) negli ultimi 30 giorni
- throughput misurato in byte/s

[7] Cf. https://en.wikipedia.org/wiki/Apdex

Capitolo 4

Raccomandazioni tecniche per REST

In questo capitolo si raccolgono delle indicazioni relative alla tecnologia REST, al fine di favorire l'interoperabilità.

4.1 Raccomandazioni sul formato dei dati

4.1.1 [RAC_REST_FORMAT_001] Utilizzo oggetti JSON

Nella tecnologia REST la comunicazione DOVREBBE avvenire tramite oggetti JSON RFC 8259 con il relativo media-type application/json.

È possibile fare eccezione in presenza di specifiche in cui gli oggetti di comunicazione sono formalizzati in forma diversa da JSON (es. INSPIRE, HL7).

4.1.2 [RAC_REST_FORMAT_002] Codificare dati strutturati con oggetti JSON

I dati strutturati in formato JSON RFC 8259 DOVREBBERO essere trasferiti tramite oggetti, in modo da permettere l'estensione retrocompatibile della response con ulteriori attributi, ad esempio paginazione.

Cioè:

- il payload di una response contenente una entry ritorna un oggetto

```
{
        "given_name": "Paolo",
        "last_name": "Rossi",
        "id": 313
}
```

- il payload di una response contenente più entry ritorna un oggetto contenente una lista e non direttamente una lista.

```
{
        "items": [
                {
                        "given_name": "Carlo",
                        "family_name": "Bianchi",
                        "id": 314
                },
                {
                        "given_name": "Giuseppe",
                        "family_name": "Verdi",
                        "id": 315
                }
        ]
}
```

4.1.3 [RAC_REST_FORMAT_003] Convenzioni di rappresentazione

DEVONO usarsi le seguenti convenzioni di rappresentazione:

- I booleani non DEVONO essere null
- Gli array vuoti non DEVONO essere null, ma liste vuote, ad es. []
- Le enumeration DEVONO essere rappresentate da stringhe non nulle

4.1.4 [RAC_REST_FORMAT_004] Definire format quando si usano i tipi Number ed Integer

I numeri e gli interi DEVONO indicare la dimensione utilizzando il parametro format.

La seguente tabella - non esaustiva - elenca un set minimo di formati.

TYPE	FORMAT	VALORI AMMESSI
integer	int32	interi tra -2^31 e 2^31-1
integer	int64	interi tra -2^63 e 2^63-1
number	decimal32 / float	IEEE 754-2008/IS 60559:2011 decimale a 32 bit
number	decimal64 / double	IEEE 754-2008/IS 60559:2011 decimale a 64 bit
number	decimal128	IEEE 754-2008/IS 60559:2011 decimale a 128 bit

Le implementazioni DEVONO utilizzare il tipo più adatto.

4.1.5 [RAC_REST_FORMAT_005] Usare link relations registrate

DEVONO usarsi le specifiche indicate in IANA registered link relations[8] per rappresentare link e riferimenti a risorse HTTP esterne.

4.2 Raccomandazioni su progettazione e naming

In assenza di specifiche regole per l'API Naming (es. HL7, INSPIRE, ..) DOVREBBERO essere adottate le seguenti regole.

4.2.1 [RAC_REST_NAME_001] Uso corretto dei metodi http

I metodi HTTP DEVONO essere utilizzati rispettando la semantica indicata in RFC 7231#section-4.3.

4.2.2 [RAC_REST_NAME_002] Usare parole separate da trattino «-» per i path (kebab-case)

Nella definizione dei path si DEVE utilizzare il separatore «-» (kebab-case).

Esempio: **/tax-code/{tax_code_id}**

Il path DOVREBBE essere semplice, intuitivo e coerente.

4.2.3 [RAC_REST_NAME_003] Preferire Hyphenated-Pascal-Case per gli header HTTP

DOVREBBE preferirsi Hyphenated-Pascal-Case per gli header HTTP.

Esempio:

```
Accept-Encoding
Apply-To-Redirect-Ref
Disposition-Notification-Options
Message-ID
```

[8] Cf. https://www.iana.org/assignments/link-relations/link-relations.xhtml

4.2.4 [RAC_REST_NAME_004] Le collezioni di risorse possono usare nomi al plurale

Si consiglia di differenziare il nome delle collezioni e delle risorse. Questo permette di separare a livello di URI, endpoint che sono in larga parte funzionalmente differenti.

Esempio 1: ricerca di documenti per data in una collezione

1. Request

```
GET /documenti?data=2018-05-01 HTTP/1.1
Host: api.example
Accept: application/json
```

2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "items": [ ".." ],
  "limit": 10,
  "next_cursor": 21314123
}
```

Esempio 2: recupera un singolo documento

1. Request

```
GET /documento/21314123 HTTP/1.1
Host: api.example
Accept: application/json
```

2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 21314123,
```

```
  "title": "Atto di nascita ..."
}
```

4.2.5 [RAC_REST_NAME_005] Utilizzare Query String standardizzate

Esempio 1: La paginazione DEVE essere implementata tramite i parametri:

```
cursor, limit, offset, sort
```

Esempio 2: La ricerca, il filtering e l'embedding dei parametri DEVE essere implementata tramite i parametri:

```
q, fields, embed
```

4.2.6 [RAC_REST_NAME_006] Non passare tramite l'header Link informazioni fornite nella response JSON

Eventuali link a risorse utili al flusso applicativo DEVONO essere restituiti nel payload JSON e non nell' HTTP header Link definito in RFC 8288. Questo semplifica l'implementazione dei client. È comunque possibile usare l''HTTP header Link per passare informazioni di tipo diverso.

Nell'esempio seguente i riferimenti alla paginazione sono riportati nel payload.

Listato 5.1 Link applicativo nel payload JSON

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "items": [ ".." ],
   "_links": {
     "next": "https://api.example/rest/library/v1/books?cursor=432123"
   }
}
```

E" corretto ad esempio estendere la risposta precedente referenziando nell'HTTP header Link l'OAS3 del servizio usando il link relation service-desc definito in RFC 8631. In questo caso il link ad openapi.yaml non serve al flusso applicativo e non è utile comunicarlo nel payload JSON.

Listato 5.2 Link applicativo nel payload JSON

```
        HTTP/1.1 200 OK
        Content-Type: application/json
        Link: <https://api.example/rest/users/v1/openapi.yaml>; rel="service-desc"; title="Books API
specifications"

        {
           "items": [ ".." ],
           "_links": {
             "next": "https://api.example/rest/library/v1/books?cursor=432123"
           }
        }
```

4.2.7 [RAC_REST_NAME_007] Usare URI assoluti nei risultati

Le response DOVREBBERO restituire URI assoluti, al fine di indicare chiaramente al client l'indirizzo delle risorse di destinazione e non obbligare i client a fare «inferenza» dal contesto.

4.2.8 [RAC_REST_NAME_008] Usare lo schema Problem JSON per le risposte di errore

In caso di errori si DEVONO ritornare:

- un payload di tipo Problem definito in RFC 7807
- il media type application/problem+json
- uno status code esplicativo
- l'oggetto, possibilmente esteso

Quando si restituisce un errore è importante non esporre dati interni delle applicazioni. Per prevenire il rischio di user-enumeration, i messaggi di errore di autenticazione non devono fornire informazioni sull'esistenza o meno dell'utenza.

Dopo aver validato il contenuto delle richieste si DEVE ritornare:

- HTTP status 415 Unsupported Media Type se il valore in HTTP header Content-Type non è supportato;
- HTTP status 400 Bad Request o HTTP status 404 Not Found se si ipotizza che la richiesta sia malevola;
- HTTP status 422 Unprocessable Entity se la representation contenuta nella richiesta è sintatticamente corretta ma semanticamente non processabile.

4.2.9 [RAC_REST_NAME_009] Ottimizzare l'uso della banda e migliorare la responsività

Si DOVREBBERO utilizzare:

- tecniche di compressione;
- paginazione;
- un filtro sugli attributi necessari;
- le specifiche di optimistic locking (HTTP header ETag , if-(none-)match) RFC 7232.

È possibile ridurre l'uso della banda e velocizzare le richieste filtrando i campi delle risorse restituite.

Esempio 1: Non filtrato

1. Request

```
GET /resources/123 HTTP/1.1
Host: api.example
```

2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{

"id": "cddd5e44-dae0-11e5-8c01-63ed66ab2da5",
"name": "Mario Rossi",
"address": "via del Corso, Roma, Lazio, Italia",
"birthday": "1984-09-13",
"partner": {
        "id": "1fb43648-dae1-11e5-aa01-1fbc3abb1cd0",
        "name": "Maria Rossi",
```

```
        "address": "via del Corso, Roma, Lazio, Italia",
        "birthday": "1988-04-07"
        }
}
```

Esempio 2: Filtrato

1. Request

```
GET /resources/123?fields=(name,partner(name)) HTTP/1.1
Host: api.example
```

2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
"name": "Mario Rossi",
"partner": {
        "name": "Maria Rossi"
        }
}
```

Si DOVREBBE effettuare la Resource Expansion per ritornare risorse correlate tra loro, in modo da ridurre il numero di richieste.In tal caso va usato:

- il parametro embed utilizzando lo stesso formato dei campi per il filtering;
- l'attributo _embedded contenente le entry espanse.

Esempio 3: Resource Expansion, utile a ritornare i dati di una persona associati ad un codice fiscale.

1. Request

```
GET /tax_code/MRORSS12T05E472W?embed=(person) HTTP/1.1
Accept: application/json
```

2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "tax_code":"MRORSS12T05E472W",
  "_embedded": {
         "person":{
               "given_name":"Mario",
               "family_name":"Rossi",
               "id":"1234-ABCD-7890"
     }
  }
}
```

4.2.10 [RAC_REST_NAME_010] Il caching http deve essere disabilitato

Il caching DOVREBBE essere disabilitato tramite HTTP header Cache-Control per evitare che delle richieste vengano inopportunamente messe in cache. Il mancato rispetto di questa raccomandazione può portare all'esposizione accidentale di dati personali.

Le API che supportano il caching DEVONO documentare le varie limitazioni e modalità di utilizzo tramite gli header definiti in RFC 7234:

- HTTP header Cache-Control
- HTTP header Vary

Eventuali conflitti nella creazione di risorse DEVONO essere gestiti tramite gli header:

- HTTP header ETag
- HTTP header If-Match
- HTTP header If-None-Match

che contengono un identificatore opaco che il server è in grado di associare univocamente alla versione della risorsa erogata. Si veda RFC 7232#section-2.3 per ulteriori informazioni su come implementare questi header.

4.2.11 [RAC_REST_NAME_011] Esporre lo stato del servizio

L'API DEVE esporre lo stato del servizio al path `/status` e ritornare un oggetto con media-type application/problem+json (RFC 7807). Se il servizio funziona correttamente, l'API ritorna HTTP status 200 OK.

Segue un esempio di specifica del path in formato OpenAPI 3.

ESEMPIO: Esposizione stato del servizio

```
openapi: 3.0.2
...
paths:
...
/status:
get:
summary: Ritorna lo stato dell'applicazione.
tags:
- public
description: \|
Ritorna lo stato dell'applicazione in formato problem+json
responses:
'200':
content:
application/problem+json:
schema:
$ref: '#/components/schemas/Problem'
description: \|
Il servizio funziona correttamente.
default:
content:
application/problem+json:
```

```
          schema:
          $ref: '#/components/schemas/Problem'
          description: \|
          Il server ha riscontrato un problema.
```

Capitolo 5

Raccomandazioni tecniche per SOAP

In questo capitolo si raccolgono delle indicazioni per la tecnologia SOAP, al fine di favorire l'interoperabilità.

5.1 Raccomandazioni globali

5.1.1 [RAC_SOAP_001] Le API SOAP DEVONO rispettare il WS-I Basic Profile versione 2.0

Questo profilo è definito dal WS-I (Web Services Interoperability Organization), ora confluito in OASIS. Questa specifica è implementata dai framework più diffusi.

5.1.2 [RAC_SOAP_002] Utilizzo di camelCase e PascalCase

Per i nomi dei servizi si DOVREBBE utilizzare PascalCase.

Per le operazioni implementate e gli argomenti si DOVREBBE utilizzare il camelCase.

5.1.3 [RAC_SOAP_003] Unicità dei namespace e utilizzo di pattern fissi

All'interno del WSDL DOVREBBE essere presente un namespace unico.

5.1.4 [RAC_SOAP_004] Esporre lo stato del servizio

L'API DEVE includere un metodo `echo` per restituire lo stato della stessa.

ESEMPIO: Esposizione stato del servizio

```
        <?xml version="1.0" encoding="UTF-8"?>
        <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ws="http://www.example.com/webservice"
targetNamespace="http://www.example.com/webservice">
          <wsdl:types>
            <xs:schema>
              <xs:complexType name="userDefinedFault">
                <xs:sequence>
                  <xs:element name="errorCode" type="xs:int"/>
                  <xs:element name="detail" type="xs:string"/>
```

```
            <xs:element name="message" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:schema>
    </wsdl:types>
    <wsdl:message name="requestEcho">
      <wsdl:part name="msg" type="xs:string"/>
    </wsdl:message>
    <wsdl:message name="responseEcho">
      <wsdl:part name="msg" type="xs:string"/>
    </wsdl:message>
    <wsdl:message name="UserDefinedException">
      <wsdl:part name="fault" type="userDefinedFault"/>
    </wsdl:message>
    <wsdl:portType name="portType">
      <wsdl:operation name="echo">
        <wsdl:input message="ws:requestEcho"/>
        <wsdl:output message="ws:responseEcho"/>
        <wsdl:fault name="fault" message="ws:UserDefinedException"/>
      </wsdl:operation>
    </wsdl:portType>
  </wsdl:definitions>
```

Capitolo 6

Gestione degli allegati

Tra i parametri o i valori di ritorno di una interfaccia di servizio può esserci la presenza di allegati, dove per allegato si intende un contenuto binario, o la cui struttura comunque non è definita direttamente dall'interfaccia di servizio (e.g., un file XML all'interno di un messaggio di SOAP in cui lo schema che definisce il messaggio SOAP non specifica anche la struttura dell'XML allegato). In generale, un allegato può essere passato o ricevuto nelle seguenti forme:

- Codificato in modo da essere rappresentabile con un set predefinito di caratteri. Il caso più comune è quello della codifica Base64.
- Come URL ad una risorsa esterna o in ogni caso come endpoint di accesso ad una risorsa.
- Nel suo formato binario originale.

Nei primi due casi, l'allegato fa parte del contenuto XML o JSON del messaggio, mentre nell'ultimo caso si ricorre a risposte di tipo multipart.

Ognuna di queste modalità presenta vantaggi e svantaggi. L'utilizzo di codifiche come Base64 oppure di URL ha il vantaggio di potere inserire un allegato all'interno del contenuto principale del messaggio. Si noti come anche nel caso di XML l'utilizzo di formati binari all'interno dei campi CDATA sia sconsigliato poiché esiste il rischio che il contenuto binario possa talvolta chiudere il campo CDATA stesso. D'altro canto, l'utilizzo di codifiche comporta un incremento nella banda richiesta poiché l'occupazione dei dati non è, in generale, ottimale. L'utilizzo di URL comporta invece un potenziale rischio potenziale poiché la risorsa collegata può essere successivamente modificata o rimossa.

In ambito SOAP, la prima proposta di standard per l'invio di allegati binari è rappresentato da SWA - SOAP with Attachments, a cui il W3C ha però preferito come standard MTOM - Message Transmission Optimization Mechanism, che ha il vantaggio di estendere agli allegati i meccanismi di sicurezza quali WS-Encryption e WS-Signature. MTOM è solitamente utilizzato insieme a XOP - XML-binary Optimized Packaging quale meccanismo per fare

riferimento agli allegati all'interno del messaggio Multipart/Related. L'utilizzo di MTOM con XOP è supportato da tutti i maggiori framework per lo sviluppo di interfacce di servizio SOAP, ma meccanismi similari, sempre basati su XOP, sono supportati anche dai maggiori framework per lo sviluppo di interfacce di servizio REST.

L'utilizzo di un approccio o di un altro dipende fortemente dallo scenario applicativo. Possono essere utilizzate seguenti regole:

- L'invio di allegati binari corrispondenti a file fa preferire solitamente l'invio di dati in formato binario, quindi mediante MTOM/XOP.
- L'utilizzo di Base64 è consigliato per l'invio di allegati di dimensioni ridotte quali ad esempio firme digitali o codici di controllo.
- L'utilizzo di URL può essere considerato nel caso in cui gli allegati siano di dimensioni tali da rendere il trasferimento via rete oneroso, a patto che si possa assicurare la persistenza della risorsa (in termini temporali) e che questa non venga modificata (a tal fine è possibile utilizzare tecniche ad esempio di hashing). Quest'ultimo caso richiede quindi solitamente trust tra fruitore ed erogatore.