3/10/2021

**Creating, Updating and Deleting Resources**

**Creating Resources - Jharna**

A resource can be created by sending a post request to a URL that represents a collection of resources. The request MUST include:

* A single resource object as primary data. The resource object MUST contain:
  + A type member.

If a relationship is provided in the relationships member of the resource object, its value MUST be:

* A relationship object with a data member. The value of this key represents the linkage the new resource is to have.

Client- Generated IDs

A server MAY accept a client generated ID along with a request to create a resource. An ID MUST be:

* Specified with an ID key
* The value of the ID MUST be a universally unique identifier (UUID) also known as a Globally Unique IDentifier (GUI). A UUID is 128 bits long and requires NO central registration process.

A server MUST return 403 Forbidden in response to an unsupported request to create a resource with a client- generated ID.

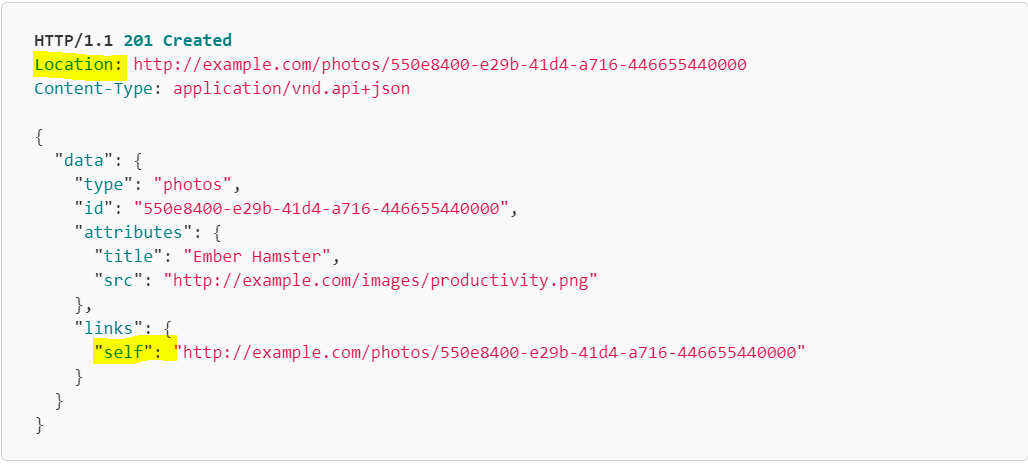
Responses

201 Created

If a POST request did not include a Client Generated ID and the requested resource has been created successfully, the server MUST return a 201 Created status code. The response should include:

* Location header identifying the location of the newly created resource.
* MUST include a document that contains the primary resource created.

(If the resource object returned by the response contains a self key in its links member and a Location header is provided, the value of the self member MUST match the value of the Location header.



202 Accepted

If a request to create a resource has been accepted for processing, but the processing has not been completed by the time the server responds, the server MUST return a 202 Accepted status code.

204 No Content

If a POST request did include a Client- Generated ID and the requested resource has been created successfully, the server MUST return either a 201 created status code and response document or a 204 No Content status code with no response document.

403 Forbidden

A server MAY return 403 Forbidden in response to an unsupported request to create a resource.

404 Not Found

A server MUST return 404 Not Found when processing a request that references a related resource that does not exist.

409 Conflict

A server MUST return 409 Conflict when processing a POST request to create a resource with a client- generated ID that already exists.

A server MUST return 409 Conflict when processing a POST request in which the resource object’s type is not among the type(s) that constitute the collection represented by the endpoint.

A server SHOULD include error details and provide enough information to recognize the source of the conflict.

Other Responses

A server MAY respond with other HTTP status codes.

A server MAY include error details with error responses.

A server MUST prepare responses, and a client MUST interpret responses, in accordance with HTTP semantics.

**Updating Resources - Markus**

To update a resource you send a PATCH request to the URL that has the resource.

The URL for a resource can be obtained in the self link of the resource object. Alternatively, when a GET request returns a single [resource object](https://jsonapi.org/format/#document-resource-objects) as primary data, the same request URL can be used for updates.

The PATCH request **MUST** include a single [resource object](https://jsonapi.org/format/#document-resource-objects) as primary data. The [resource object](https://jsonapi.org/format/#document-resource-objects) **MUST** contain type and id members.

For example:

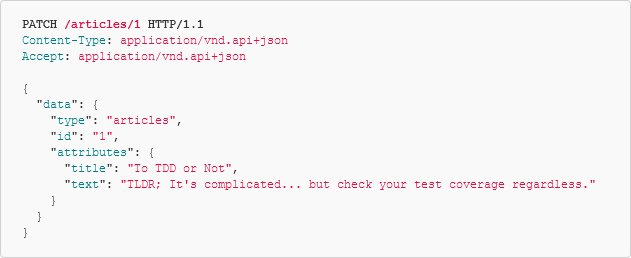


#### **Updating a Resource’s Attributes**

Any or all of a resource’s [attributes](https://jsonapi.org/format/#document-resource-object-attributes) **MAY** be included in the resource object included in a PATCH request.

If a request does not include all of the [attributes](https://jsonapi.org/format/#document-resource-object-attributes) for a resource, the server **MUST** interpret the missing [attributes](https://jsonapi.org/format/#document-resource-object-attributes) as if they were included with their current values. The server **MUST NOT** interpret missing attributes as null values.

For example, the following PATCH request is interpreted as a request to update only the title and text attributes of an article:



#### **Updating a Resource’s Relationships**

Any or all of a resource’s [relationships](https://jsonapi.org/format/#document-resource-object-relationships) **MAY** be included in the resource object included in a PATCH request.

If a request does not include all of the [relationships](https://jsonapi.org/format/#document-resource-object-relationships) for a resource, the server **MUST** interpret the missing [relationships](https://jsonapi.org/format/#document-resource-object-relationships) as if they were included with their current values. It **MUST NOT** interpret them as null or empty values.

If a relationship is provided in the relationships member of a resource object in a PATCH request, its value **MUST** be a relationship object with a data member. The relationship’s value will be replaced with the value specified in this member.

For instance, the following PATCH request will update the author relationship of an article:



Likewise, the following PATCH request performs a complete replacement of the tags for an article:



A server **MAY** reject an attempt to do a full replacement of a to-many relationship. In such a case, the server **MUST** reject the entire update, and return a 403 Forbidden response.

Note: Since full replacement may be a very dangerous operation, a server may choose to disallow it. For example, a server may reject full replacement if it has not provided the client with the full list of associated objects, and does not want to allow deletion of records the client has not seen.

#### **Responses**

##### **202 Accepted**

If an update request has been accepted for processing, but the processing has not been completed by the time the server responds, the server **MUST** return a 202 Accepted status code.

##### **200 OK**

If a server accepts an update but also changes the resource(s) in ways other than those specified by the request (for example, updating the updated-at attribute or a computed sha), it **MUST** return a 200 OK response. The response document **MUST** include a representation of the updated resource(s) as if a GET request was made to the request URL.

A server **MUST** return a 200 OK status code if an update is successful, the client’s current fields remain up to date, and the server responds only with top-level [meta](https://jsonapi.org/format/#document-meta) data. In this case the server **MUST NOT** include a representation of the updated resource(s).

##### **204 No Content**

If an update is successful and the server doesn’t update any fields besides those provided, the server **MUST** return either a 200 OK status code and response document (as described above) or a 204 No Content status code with no response document.

##### **403 Forbidden**

A server **MUST** return 403 Forbidden in response to an unsupported request to update a resource or relationship.

##### **404 Not Found**

A server **MUST** return 404 Not Found when processing a request to modify a resource that does not exist.

A server **MUST** return 404 Not Found when processing a request that references a related resource that does not exist.

##### **409 Conflict**

A server **MAY** return 409 Conflict when processing a PATCH request to update a resource if that update would violate other server-enforced constraints (such as a uniqueness constraint on a property other than id).

A server **MUST** return 409 Conflict when processing a PATCH request in which the resource object’s type and id do not match the server’s endpoint.

A server **SHOULD** include error details and provide enough information to recognize the source of the conflict.

##### **Other Responses**

A server **MAY** respond with other HTTP status codes.

A server **MAY** include [error details](https://jsonapi.org/format/#errors) with error responses.

A server **MUST** prepare responses, and a client **MUST** interpret responses, in accordance with [HTTP semantics](http://tools.ietf.org/html/rfc7231).

**Updating Relationships - Lennart**

JSON:API Allows you to update relationships independently at URLs from relationship links.

Note: Relationships are updated without exposing the underlying server semantics, such as foreign keys. Furthermore, relationships can be updated without necessarily affecting the related resources. For example, if an article has many authors, it is possible to remove one of the authors from the article without deleting the person itself. Similarly, if an article has many tags, it is possible to add or remove tags. Under the hood on the server, the first of these examples might be implemented with a foreign key, while the second could be implemented with a join table, but the JSON:API protocol would be the same in both cases.

Note: A server may choose to delete the underlying resource if a relationship is deleted (as a garbage collection measure).

#### Updating To-One Relationships

A server MUST respond to PATCH requests to a URL from a to-one [relationship link](https://jsonapi.org/format/#document-resource-object-relationships) as described below.

The PATCH request MUST include a top-level member named data containing one of:

* a resource identifier object corresponding to the new related resource.
* null, to remove the relationship.

For example, the following request updates the author of an article:



And the following request clears the author of the same article:



If the relationship is updated successfully then the server MUST return a successful response.

#### Updating To-Many Relationships

A server MUST respond to PATCH, POST, and DELETE requests to a URL from a to-many relationship link as described below.

For all request types, the body MUST contain a data member whose value is an empty array or an array of resource identifier objects.

If a client makes a PATCH request to a URL from a to-many relationship link, the server MUST either completely replace every member of the relationship, return an appropriate error response if some resources can not be found or accessed, or return a 403 Forbidden response if complete replacement is not allowed by the server.

For example, the following request replaces every tag for an article:



And the following request clears every tag for an article:



If a client makes a POST request to a URL from a [relationship link](https://jsonapi.org/format/#document-resource-object-relationships), the server MUST add the specified members to the relationship unless they are already present. If a given type and id is already in the relationship, the server MUST NOT add it again.

Note: This matches the semantics of databases that use foreign keys for has-many relationships. Document-based storage should check the has-many relationship before appending to avoid duplicates.

If all of the specified resources can be added to, or are already present in, the relationship then the server MUST return a successful response.

Note: This approach ensures that a request is successful if the server’s state matches the requested state, and helps avoid pointless race conditions caused by multiple clients making the same changes to a relationship.

In the following example, the comment with ID 123 is added to the list of comments for the article with ID 1:



If the client makes a DELETE request to a URL from a [relationship link](https://jsonapi.org/format/#document-resource-object-relationships) the server MUST delete the specified members from the relationship or return a 403 Forbidden response. If all of the specified resources are able to be removed from, or are already missing from, the relationship then the server MUST return a successful response.

Note: As described above for POST requests, this approach helps avoid pointless race conditions between multiple clients making the same changes.

Relationship members are specified in the same way as in the POST request.

In the following example, comments with IDs of 12 and 13 are removed from the list of comments for the article with ID 1:



Note: RFC 7231 specifies that a DELETE request may include a body, but that a server may reject the request. This spec defines the semantics of a server, and we are defining its semantics for JSON:API.

#### **Responses**

##### 202 Accepted

If a relationship update request has been accepted for processing, but the processing has not been completed by the time the server responds, the server MUST return a 202 Accepted status code.

##### 204 No Content

A server MUST return a 204 No Content status code if an update is successful and the representation of the resource in the request matches the result.

Note: This is the appropriate response to a POST request sent to a URL from a to-many [relationship link](https://jsonapi.org/format/#document-resource-object-relationships) when that relationship already exists. It is also the appropriate response to a DELETE request sent to a URL from a to-many [relationship link](https://jsonapi.org/format/#document-resource-object-relationships) when that relationship does not exist.

##### 200 OK

If a server accepts an update but also changes the targeted relationship(s) in other ways than those specified by the request, it MUST return a 200 OK response. The response document MUST include a representation of the updated relationship(s).

A server MUST return a 200 OK status code if an update is successful, the client’s current data remain up to date, and the server responds only with top-level meta data. In this case the server MUST NOT include a representation of the updated relationship(s).

##### 403 Forbidden

A server MUST return 403 Forbidden in response to an unsupported request to update a relationship.

##### Other Responses

A server MAY respond with other HTTP status codes.

A server MAY include error details with error responses.

A server MUST prepare responses, and a client MUST interpret responses, in accordance with HTTP semantics.

**DELETING RESOURCES-Lydia**

An individual resource can be deleted by making a DELETE request to the resource’s URL:

The syntax for a delete request is DELETE /file.html HTTP/1.1

If a DELETE method is successfully applied, there are several response status codes possible:

A **202 (Accepted)** status code if the action will likely succeed but has not yet been enacted.

A **204 (No Content)** status code if the action has been enacted and no further information is to be supplied.

A **200 (OK)** status code if the action has been enacted and the response message includes a representation describing the status (meta data)

A **404 (Not found)** may be returned if a deletion request fails due to the resource not existing

**Other Responses**

-A server MAY respond with other HTTP status codes.

-A server MAY include error details with error responses.

-A server MUST prepare responses, and a client MUST interpret responses, in accordance with HTTP semantics.

**Query/ Error: -Shanta**

Query Parameters:

· -It must have a member name with the other requirements.

· It must contain at least non a-z character(U+00061 to U+0007A)

· [ for example, (U+00061)-> here we have only one character and the rest are sign & numbers.]

· It is the best practice that a U+002D HYPHEN-MINUS, “-“, U+005F LOW LINE, “\_”, or capital letter is used (e.g. camelCasing).

· If the naming conventions are not followed, then the server will return a “400 Bad Request”.

# 

# 

# 

# 

# **Errors:**

There are two different types of errors :- **Processing Errors & Error Objects**.

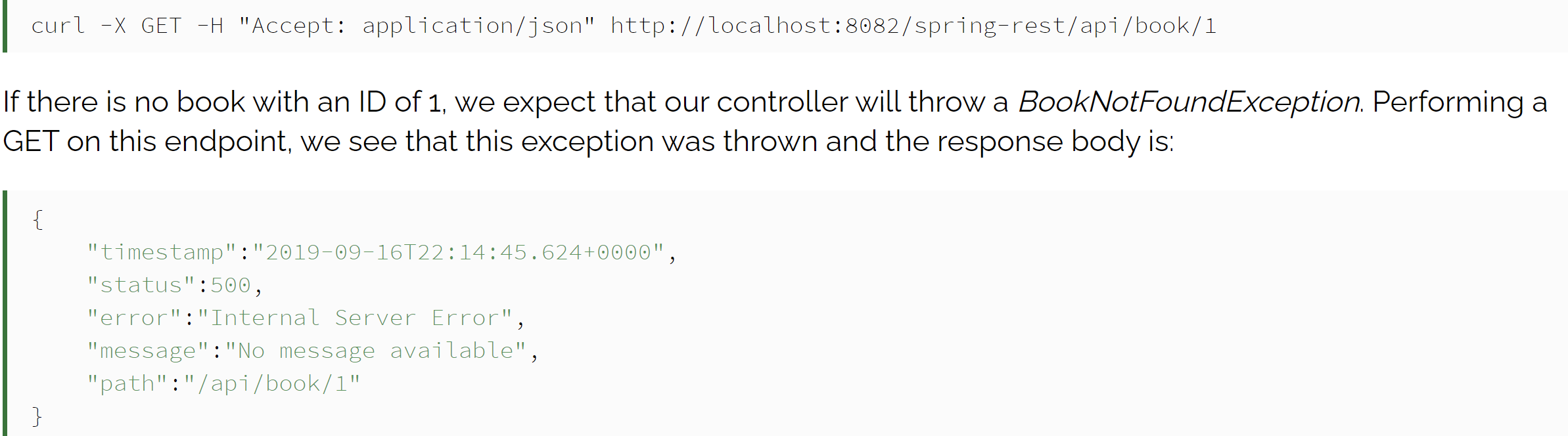
# **Processing Errors**

A server in general chooses to stop processing as soon as a problem is encountered, or else it continues processing and encounters multiple problems. For example, a server processes multiple attributes and then returns multiple validation problems in a single response.

When a server encounters multiple problems for a single request, the most generally applicable HTTP error code needs to be used in the response. For instance,

-400 Bad Request might be appropriate for multiple 4xx errors.

-500 Internal Server Error might be appropriate for multiple 5xx errors.

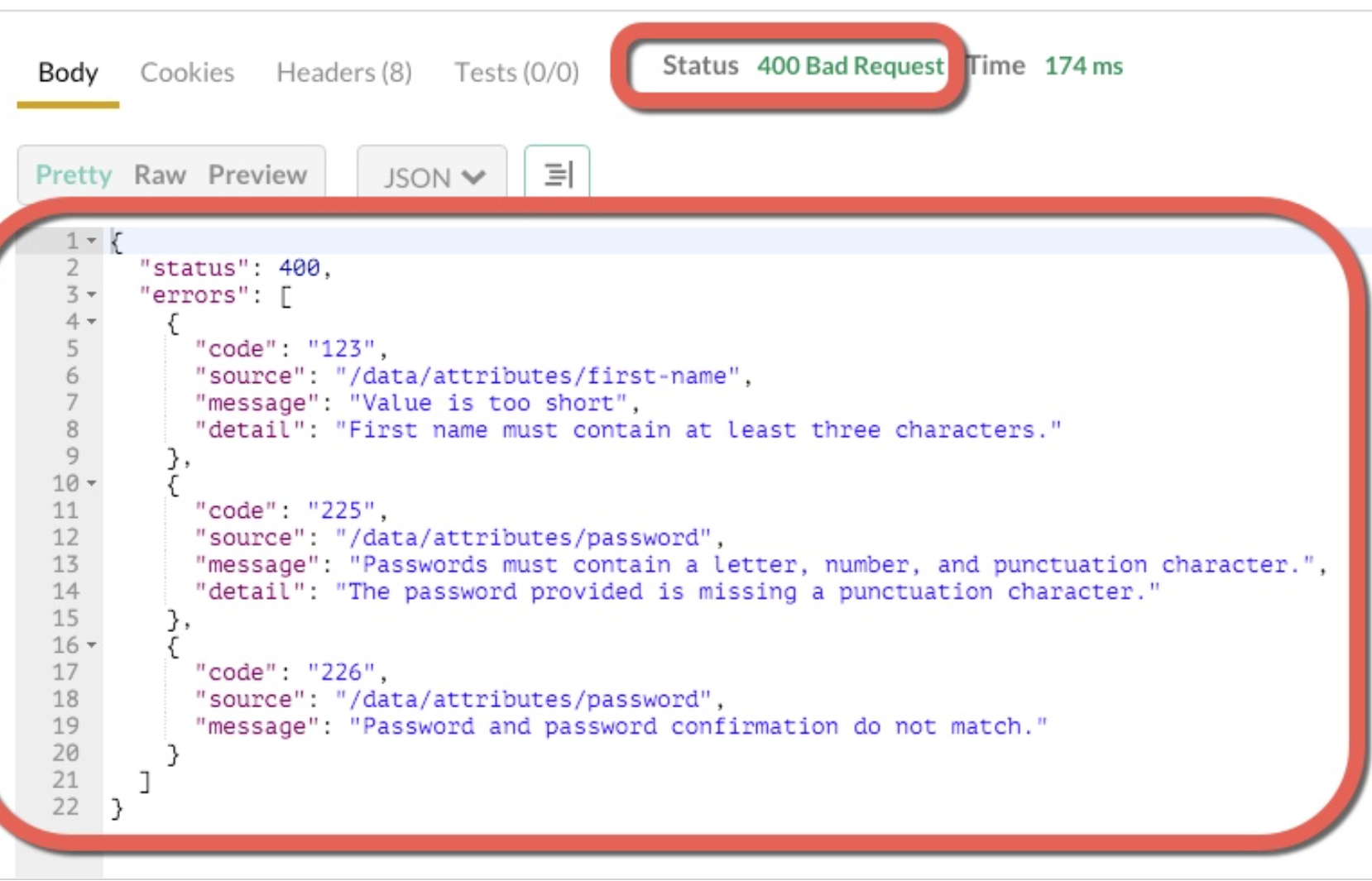


# **Error Objects**

Error objects provide additional information about problems encountered while performing an operation. As an array keyed by errors , Error objects must be returned in the top level of a JSON:API document,

An error object will have the following members:

* id: a unique identifier for this particular occurrence of the problem.
* links: a [links object](https://jsonapi.org/format/#document-links) containing the following members:
  + about: a [link](https://jsonapi.org/format/#document-links) that leads to further details about this particular occurrence of the problem.
* status: the HTTP status code applicable to this problem, expressed as a string value.
* code: an application-specific error code, expressed as a string value.
* title: a short, human-readable summary of the problem that must not change from occurrence to occurrence of the problem, except for purposes of localization.
* detail: a human-readable explanation specific to this occurrence of the problem. Like the title, this field’s value can be localized.
* source: an object containing references to the source of the error, optionally including any of the following members:
  + pointer: a JSON Pointer [[RFC6901](https://tools.ietf.org/html/rfc6901)] to the associated entity in the request document [e.g. "/data" for a primary data object, or "/data/attributes/title" for a specific attribute]
  + parameter: a string indicating which URI query parameter caused the error.
* meta: a [meta object](https://jsonapi.org/format/#document-meta) containing non-standard meta-information about the error.



FHIR

If we take for example the healthcare FHIR standard, the error is defined there as an FHIR resource called operation outcome, here is an example of the error response:

{

"resourceType": "OperationOutcome",

"id": "exception",

"text": {

"status": "additional",

"div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">\n<p>SQL Link Communication Error (dbx = 34234)</p>\n</div>"

},

"issue": [

{

"severity": "error",

"code": "exception",

"details": {

"text": "SQL Link Communication Error (dbx = 34234)"

}

}

]

}

The concept is very good, the interface (<https://www.hl7.org/fhir/operationoutcome.html>) is very generic and can be used for many responses, even successful ones. However in order to use it, you need to know the FHIR standard from A-Z and for some, the healthcare concepts are "a bit" overkill.

###### Google

Some companies like Google is using the following error response for their APIs:

{

"error": {

"errors": [

{

"domain": "global",

"reason": "invalidParameter",

"message": "Invalid string value: 'asdf'. Allowed values: [mostpopular]",

"locationType": "parameter",

"location": "chart"

}

],

"code": 400,

"message": "Invalid string value: 'asdf'. Allowed values: [mostpopular]"

}

}

<https://developers.google.com/doubleclick-search/v2/standard-error-responses>

I just think that the JSON structure is a bit strange - errors array inside an error object?

###### Facebook

For Facebook the error response is different as well:

{

"error": {

"message": "Message describing the error",

"type": "OAuthException",

"code": 190,

"error\_subcode": 460,

"error\_user\_title": "A title",

"error\_user\_msg": "A message",

"fbtrace\_id": "EJplcsCHuLu"

}

}

###### **Spotify**

{

"error" : {

"status" : 502,

"message" : "Bad gateway."

}

}

There are other examples out there but I think that everyone will agree that most of the error code structures are different. How you reference links, what error code you generate, and how to display those codes is subject to change from company to company.

However, there has been headway to standardize these approaches; the IETF recently published RFC 7807, which outlines how to use a JSON object as a way to model problem details within HTTP response.

##### RFC 7807 To The Rescue

This document defines a "problem detail" as a way to carry machine-readable details of errors in an HTTP response to avoid the need to define new error response formats for HTTP APIs.

By providing more specific machine-readable messages with an error response, the API clients can react to errors more effectively and eventually it makes the API services much more reliable from the [REST API testing](https://www.restcase.com/) perspective and the clients as well.

In general, the goal of error responses is to create a source of information to not only inform the user of a problem but of the solution to that problem as well. Simply stating a problem does nothing to fix it – and the same is true of API failures.

RFC 7807 provides a standard format for returning problem details from HTTP APIs. In particular, it specifies the following:

* Error responses MUST use standard HTTP status codes in the 400 or 500 range to detail the general category of error.
* Error responses will be of the Content-Type application/problem, appending a serialization format of either json or xml: application/problem+json, application/problem+xml.
* Error responses will have each of the following keys:
  + detail (string) - A human-readable description of the specific error.
  + type (string) - a URL to a document describing the error condition (optional, and "about:blank" is assumed if none is provided; should resolve to a human-readable document).
  + title (string) - A short, human-readable title for the general error type; the title should not change for given types.
  + status (number) - Conveying the HTTP status code; this is so that all information is in one place, but also to correct for changes in the status code due to the usage of proxy servers. The status member, if present, is only advisory as generators MUST use the same status code in the actual HTTP response to assure that generic HTTP software that does not understand this format still behaves correctly.
  + instance (string) - This optional key may be present, with a unique URI for the specific error; this will often point to an error log for that specific response.

##### {

##### "type": "https://example.net/validation-error",

##### "title": "Your request parameters didn't validate.",

##### "invalid-params": [ {

##### "name": "age",

##### "reason": "must be a positive integer"

##### },

##### {

##### "name": "color",

##### "reason": "must be 'green', 'red' or 'blue'"}

##### ]

##### }