Standard JSON API response format?

611

361

Do standards or best practices exist for structuring JSON responses from an API? Obviously every application's data is different, so that much I'm not concerned with, but rather the "response boilerplate", if you will. An example of what I mean:

Successful request:

```
{
  "success": true,
  "payload": {
    /* Application-specific data would go here. */
  }
}
```

Failed request:

```
{
  "success": false,
  "payload": {
    /* Application-specific data would go here. */
  },
  "error": {
    "code": 123,
    "message": "An error occurred!"
  }
}
```

json request response

edited May 25 '16 at 19:09
Laurel
4,847 10 22 40

asked Oct 9 '12 at 18:43
FtDRbwLXw6
16k 12 52 99

response format. Yours actually looks quite nice, and looks worth using if you don't find a standard. It's a shame that the answers provided don't actually address the question. – Alex Oct 11 '12 at 21:05

12 @Alex unfortunately, that's because no matter where you go, there is *no* standard. Not only within JSON itself, but in terms of how to use it for RESTful applications, or anything else of the sort. Everybody does it differently. You can feel free to follow best-practices (HTTP-responses, meaningful package-structure, an eye towards structuring your data for consumption by your system), but ***everybody*** who is a major distributor is doing at least one thing different than the others... There is no standard, and there won't likely be one, so build something solid, and build it to fit you. – Norguard Oct 12 '12 at 2:41

5 @Norguard there are standards (see my answer). In fact *The nice thing about standards is that you have so many to choose from.* - Andrew Tanenbaum – Adam Gent Jan 26 '13 at 16:16

13 Answers

563

Yes there are a couple of standards (albeit some liberties on the definition of standard) that have emerged:

1. JSON API - JSON API covers creating and updating resources as well, not just responses.
2. JSend - Simple and probably what you are already doing.
3. OData JSON Protocol - Very complicated.
4. HAL - Like OData but aiming to be HATEOAS like.

There are also JSON API description formats:

- Swagger
  - JSON Schema (used by swagger but you could use it stand alone)
- WADL in JSON
- RAML
- HAL because HATEOAS in theory is self describing.

edited Jan 15 at 18:22 xzilla

answered Jan 26 '13 at 16:12 Adam Gent

12 There is also jsonapi.org – Jafin Nov 13 '13 at 3:48

7 For error responses specifically I also like the Problem Details for HTTP APIs RFC draft. – Pieter Ennes Feb 13 '14 at 12:36

1 Maybe you want to add code.google.com/p/json-service to the description format list? – emilesilvis Aug 15 '14 at 15:43

1 I think the "A recommended standard for Rails" label is a bit of an overstatement - this is just one programmer's solution. Not sure what makes it a "recommended standard" (specially if you look at the gem's popularity - doesn't look like that many people is using this at all)? I personally don't think most Rails programmers would recommend this solution because of using response body instead of HTTP headers for status – Iwo Dziechciarow Sep 4 '14 at 21:00

Google JSON guide

165

Success response return `data`

```
{
  "data": {
    "id": 1001,
    "name": "Wing"
  }
}
```

Error response return `error`

```
{
  "error": {
    "code": 404,
    "message": "ID not found"
  }
}
```

and if your client is JS, you can using `if ("error" in response) {}` to check if there is error.

edited Feb 27 '17 at 9:55 answered May 17 '14 at 7:46

1 Im not sure if you can handle this from a Server Side JSON API like PlayJson, either way it doesn't matter. @Steely your links are broken – Rhys Bradbury May 26 '16 at 10:26

3 What about errors that need to provide a list of failures (like validation problems)? – Xeoncross Dec 21 '17 at 2:08

1 @Xeoncross click the link on the word `error` , Google's page gives an example of this – M. I. Wright Dec 22 '17 at 19:00

@Xeoncross You can return a list of failures using error.errors[], defined as: "Container for any additional information regarding the error. If the service returns multiple errors, each element in the errors array represents a different error." Perhaps the top level error would mention "Request failed input validation" and the errors[] array would have one entry for each specific validation failure that occurred. – James Daily Jan 16 at 20:48

104

I guess a defacto standard has not really emerged (and may never). But regardless, here is my take:

Successful request:

```
{
  "status": "success",
  "data": {
    /* Application-specific data would go here. */
  },
  "message": null /* Or optional success message */
}
```

Failed request:

```
{
  "status": "error",
  "data": null, /* or optional error payload */
  "message": "Error xyz has occurred"
}
```

Advantage: Same top level elements in both success and error cases

Disadvantage: No error code, but if you want, you can either change status to be a (success or failure) code, -or- you can add another top level item named "code".

3 yes this is right way if you are using POJO for json parsing! when we are using POJOs we need static, non dynamic json format! – LOG_TAG Apr 10 '14 at 11:04

Simple and to the point. Better than jsend in my opinion because jsend distinguishes error from fail. – Josue Alexander Ibarra Apr 11 '14 at 22:11

1 I use this pattern too but with a field called `messages` which is an **array of messages** instead of a single string. – StockBreak May 29 '15 at 12:45

3 The answer is almost a copy of well documented JSend which is simple and very useful. They provided third status `fail` for typical validation problems, while `error` is used only with fatals like db errors. – s3m3n Jan 27 '16 at 13:44

for the success: if it has `200` in the headers why do you even need a `status` field? just return the data object straight. You know this can cause additional pain with typed FE languages like TypeScript. – Deniss M. Oct 8 '18 at 11:55

78

Assuming you question is about REST webservices design and more precisely concerning success/error.

I think there are 3 different types of design.

1. Use **only HTTP Status code** to indicate if there was an error and try to limit yourself to the standard ones (usually it should suffice).
   - Pros: It is a standard independent of your api.
   - Cons: Less information on what really happened.
2. Use **HTTP Status + json body** (even if it is an error). Define a uniform structure for errors (ex: code, message, reason, type, etc) and use it for errors, if it is a success then just return the expected json response.
   - Pros: Still standard as you use the existing HTTP status codes and you return a json describing the error (you provide more information on what happened).
   - Cons: The output json will vary depending if it is a error or success.
3. **Forget the http status** (ex: always status 200), always use json and add at the root of the response a boolean responseValid and a error object (code,message,etc) that will be populated if it is an error otherwise the other fields (success) are populated.
   - Pros: The client deals only with the body of the response that is a json string and ignores the status(?).
   - Cons: The less standard.

It's up to you to choose :)

EDIT:

- Solution 2 is the hardest to implement but the advantage is that you can nicely handle exceptions and not only business errors, initial effort is more important but you win on the long term.
- Solution 3 is the easy to implement on both, server side and client but it's not so nice as you will have to encapsulate the objects you want to return in a response object containing also the responseValid + error.

edited Mar 24 '17 at 4:56

answered Oct 9 '12 at 19:49

eugen
5,327 1 22 25

2 You say that I should "Define a uniform structure for errors" and other similar suggestions, but this is precisely what I'm asking about. I guess the answer is turning out to be that "no, there is no standard or best practices with regards to this structure." – FtDRbwLXw6 Oct 9 '12 at 21:31

7 For the record: the HTTP status code is not a header. – pepkin88 Mar 25 '15 at 14:56

3 "the response will not be json but html." wrong! html has nothing to do with error handling. the response can be whatever content-type you support. – oligofren Oct 8 '15 at 10:47

@pepkin88, what is it then? – アレックス Apr 14 '17 at 21:48

1 @アレックス HTTP status code is a 3-digit code in the status line of the header of an HTTP response. Following that line are header fields, colloquially also called headers. – pepkin88 Apr 15 '17 at 20:08

19

I will not be as arrogant to claim that this is a standard so I will use the "I prefer" form.

I prefer terse response (when requesting a list of /articles I want a JSON array of articles).

In my designs I use HTTP for status report, a **200** returns just the payload.

400 returns a message of what was wrong with request:

```
{"message" : "Missing parameter: 'param'"}
```

```
{"message" : "Could not connect to data store."}
```

From what I've seen quite a few REST-ish frameworks tend to be along these lines.

Rationale:

JSON is supposed to be a **payload** format, it's not a session protocol. The whole idea of verbose session-ish payloads comes from the XML/SOAP world and various misguided choices that created those bloated designs. After we realized all of it was a massive headache, the whole point of REST/JSON was to KISS it, and adhere to HTTP. I don't think that there is anything remotely *standard* in either JSend and especially not with the more verbose among them. XHR will react to HTTP response, if you use jQuery for your AJAX (like most do) you can use `try` / `catch` and `done()` / `fail()` callbacks to capture errors. I can't see how encapsulating status reports in JSON is any more useful than that.

edited Apr 16 '14 at 5:40

community wiki
2 revs
Bojan Markovic

"JSON is a payload format..." – Dirk Bester Apr 16 '14 at 0:07

2 "JSON is a payload format...". No, JSON is a data serialization format. You can use it to transmit anything you want, including session protocols or just simple payloads. Your KISS comments are on target though and independent of JSON. Better to keep the JSON focused on what it is (success data or failure reason data as you describe) than pollute it with some mishmash of both that constantly has to be composed and later stripped out. Then you can go all the way and store your JSON data as is in Couchbase and return it as is to the application. – Dirk Bester Apr 16 '14 at 0:32

1 Perhaps I should have formulated it as "supposed to be a payload format", but apart from that, I stand by my comment. You could put session/error data as attributes of *body* tag in HTML document, but that does not make it the right or sensible way to do it. – Bojan Markovic Apr 16 '14 at 5:40

17

Following is the json format instagram is using

```
{
    "meta": {
        "error_type": "OAuthException",
        "code": 400,
        "                "  "   "
```

```
        "next_max_id": "13872296"
    }
}
```

answered Mar 13 '13 at 8:19

Muhammad Amin
853 6 11

15

For what it's worth I do this differently. A successful call just has the JSON objects. I don't need a higher level JSON object that contains a success field indicating true and a payload field that has the JSON object. I just return the appropriate JSON object with a 200 or whatever is appropriate in the 200 range for the HTTP status in the header.

However, if there is an error (something in the 400 family) I return a well-formed JSON error object. For example, if the client is POSTing a User with an email address and phone number and one of these is malformed (i.e. I cannot insert it into my underlying database) I will return something like this:

```
{
  "description" : "Validation Failed"
  "errors" : [ {
    "field" : "phoneNumber",
    "message" : "Invalid phone number."
  } ],
}
```

Important bits here are that the "field" property must match the JSON field exactly that could not be validated. This allows clients to know exactly what went wrong with their request. Also, "message" is in the locale of the request. If both the "emailAddress" and "phoneNumber" were invalid then the "errors" array would contain entries for both. A 409 (Conflict) JSON response body might look like this:

```
{
  "description" : "Already Exists"
  "errors" : [ {
    "field" : "phoneNumber",
    "message" : "Phone number already exists for another user."
```

anything in the 200 range I can just return whatever is appropriate. For me it is often a HAL-like JSON object but that doesn't really matter here.

The one thing I thought about adding was a numeric error code either in the the "errors" array entries or the root of the JSON object itself. But so far we haven't needed it.

answered Apr 28 '14 at 18:45

robert_difalco
3,149 2 24 47

15

The RFC 7807: Problem Details for HTTP APIs is at the moment the closest thing we have to an official standard.

answered Apr 3 '16 at 20:39

Berislav Lopac
10.5k 4 53 66

1 3 years later... seems to be the direction to go. See also:youtu.be/vcjj5pT0bSQ?t=611 (Visual Studio .Net core Support for 7807) – edelwater Apr 21 at 21:49

9

Their is no agreement on the rest api response formats of big software giants - Google, Facebook, Twitter, Amazon and others, though many links have been provided in the answers above, where some people have tried to standardize the response format.

As needs of the API's can differ it is very difficult to get everyone on board and agree to some format. If you have millions of users using your API, why would you change your response format?

Following is my take on the response format inspired by Google, Twitter, Amazon and some posts on internet:

https://github.com/adnan-kamili/rest-api-response-format

Swagger file:

1 upvote for the envelope-free rest-api-response-format – Kerem Baydoğan Oct 27 '16 at 11:50

7

The point of JSON is that it is completely dynamic and flexible. Bend it to whatever whim you would like, because it's just a set of serialized JavaScript objects and arrays, rooted in a single node.

What the type of the rootnode is is up to you, what it contains is up to you, whether you send metadata along with the response is up to you, whether you set the mime-type to `application/json` or leave it as `text/plain` is up to you (as long as you know how to handle the edge cases).

Build a lightweight schema that you like.
Personally, I've found that analytics-tracking and mp3/ogg serving and image-gallery serving and text-messaging and network-packets for online gaming, and blog-posts and blog-comments **all** have ***very different requirements*** in terms of what is sent and what is received and how they should be consumed.

So the last thing I'd want, when doing all of that, is to try to make each one conform to the same boilerplate standard, which is based on XML2.0 or somesuch.

That said, there's a lot to be said for using schemas which make sense to ***you*** and are well thought out.
Just read some API responses, note what you like, criticize what you don't, write those criticisms down and understand why they rub you the wrong way, and then think about how to apply what you learned to what you need.

answered Oct 9 '12 at 19:06

Norguard
22.1k 4 34 43

1 Thank you for the response, but again, I'm not worried about the payloads themselves. While your examples all have very different requirements in terms of what is sent/received within the *payloads* and how those *payloads* are consumed, they all have to solve the same problems with respect to the *response itself*. Namely, they all need to determine if the request was successful. If it was, proceed with processing. If it wasn't, what went wrong. It's this boilerplate that is common to *all* API responses that I'm referring to in my question. – FtDRbwLXw6 Oct 9 '12 at 19:36

Either return a status of 200 for everything, and define yourself a specific error payload, or return a status commensurate with the error, with and/or without more details in the body of the payload (if supported). Like I said, the schema is up to you - including any meta/status information. It's a 100% blank slate to do with what you please based on your preferred style of architecture. – Norguard Oct 9 '12 at 22:10

5

JSON-RPC 2.0 defines a standard request and response format, and is a breath of fresh air after working with REST APIs.

answered Jun 11 '14 at 22:09

dnault
4,206 1 19 40

The only thing JSON-RPC_2.0 offers for exceptions is an error code? A numeric error code can not represent with any fidelity the problem that occurred. – AgilePro May 2 at 18:58

@AgilePro Agreed, a numeric error code is not very nice, and I wish the authors of the spec had allowed the `code` field to be a String. Fortunately the spec allows us to stuff whatever information we want into the error's `data` field. In my JSON-RPC projects I typically use a single numeric code to for all application-layer errors (as opposed to one of the standard protocol errors). Then I put the detailed error information (including a string code indicating the real error type) in the `data` field. – dnault May 2 at 19:36

0

The basic framework suggested looks fine, but the error object as defined is too limited. One often cannot use a single value to express the problem, and instead a chain of problems and causes is needed.

I did a little research and found that the most common format for returning error (exceptions) is a structure of this form:

```
{
   "success": false,
   "error": {
      "code": "400",
      "message": "main error message here",
      "target": "approx what the error came from",
      "details": [
         {
            "code": "23-098a",
            "message": "Disk drive has frozen up again.  It needs to be replaced",
            "target": "not sure what the target is"
         }
      ],
      "innererror": {
```

This is the format proposed by the OASIS data standard OASIS OData and seems to be the most standard option out there, however there does not seem to be high adoption rates of any standard at this point. This format is consistent with the JSON-RPC specification.

You can find the complete open source library that implements this at: Mendocino JSON Utilities. This library supports the JSON Objects as well as the exceptions.

The details are discussed in my blog post on Error Handling in JSON REST API

edited May 4 at 0:52

answered Jan 7 at 23:28

AgilePro

3,468 2 21 46

-1

Best Response for web apis that can easily understand by mobile developers.

This is for "Success" Response

```
{
   "ReturnCode":"1",
   "ReturnMsg":"Successfull Transaction",
   "ReturnValue":"",
   "Data":{
      "EmployeeName":"Admin",
      "EmployeeID":1
   }
}
```

This is for "Error" Response

```
{
    "ReturnCode": "4",
    "ReturnMsg": "Invalid Username and Password",
    "ReturnValue": "",
    "Data": {}
}
```