

1. Rest API Best Practice:

URIs as resources as nouns:

One of the most recognizable characteristics of REST is the predominant use of nouns in URIs. Restful URIs should not indicate any kind of CRUD (Create, Read, Update, and Delete) functionality. Instead, REST APIs

Forward slashes for hierarchy:

As shown in the examples above, forward slashes are conventionally used to show the hierarchy between individual resources and collections

Example: `/users/{id}` instead of `/geUser`

Punctuation for lists:

When there is no **hierarchical relationship (such as in lists)**, punctuation marks **such as the semicolon, or, more frequently, the comma** should be used.

Example: `/users/{id1},{id2}` to access multiple user resources

Query parameters where necessary:

In order to sort or filter a collection, a REST API should **allow query parameters** to be passed in the URI Example: `/users?location=USA` to find all users living in the United States

Lowercase letters and dashes:

By convention, resource names should use exclusively **lowercase letters**. Similarly, **dashes (-)** are conventionally used in place of underscores (_).

Example: `/users/{id}/pending-orders` instead of `/users/{id}/Pending_Orders`

No file extensions:

Leave file extensions (such as .xml) out of your URIs. We're sorry to say it, but they're ugly and add length to URIs. If you need to specify the format of the body, instead use the Content-Type header

Example: `/users/{id}/pending-orders` instead of `/users/{id}/pending-orders.xml`

No trailing forward slash: Similarly, in the interests of **keeping URIs clean, do not add a trailing forward slash** to the end of URIs.

Example: `/users/{id}/pending-orders` instead of `/users/{id}/pending-orders/`

3. JSON Best Practices

JSON is no doubt a flexible way to share data across systems. But that doesn't mean JSON can be created any way.

Here are some best practices which will help your consumers to use consume your output.

Enclose within DOUBLE Quotes

Always enclose the **Key : Value** pair within **double quotes**. It may be convenient (not sure how) to generate with Single quotes, but JSON parser don't like to parse JSON objects with single quotes.

For numerical Values, quotes are optional but is a good practice to enclose them in double quote.

```
{'id': '1', 'name': File} is not right ✗ {"id": 1, "name": "File"} is okay  
✓ {"id": "1", "name": "File"} is the best ✓
```

No Hyphens please

Never Never Never use Hyphens in your Key fields. It breaks python, scala parser and developers have to escape it to use those fields.

Instead of Hyphens use underscores (_). But using alllower case or camel Case is the best. See samples below.

```
{"first-name": "Rachel", "last-name": "Green"} is not right.  
✗ {"first_name": "Rachel", "last_name": "Green"} is okay  
✓ {"firstname": "Rachel", "lastname": "Green"} is okay  
✓ {"firstName": "Rachel", "lastName": "Green"} is the best. ✓
```

Always create a Root element.

Creation of Root element is optional, but it helps when you are generating complicated JSON.

```
JSON with root element{  
  "menu": [  
    {  
      "id": "1",  
      "name": "File",  
      "value": "F",  
      "popup": {  
        "menuitem": [  
          {"name": "New", "value": "1N", "onclick": "newDoc()"},  
          {"name": "Open", "value": "1O", "onclick": "openDoc()"},  
          {"name": "Close", "value": "1C", "onclick": "closeDoc()"}  
        ]  
      }  
    },  
    {  
      "id": "2",  
      "name": "Edit",  
      "value": "E",  
      "popup": {  
        "menuitem": [  

```

```

        {"name": "Undo", "value": "2U", "onclick": "undo()"},
        {"name": "Copy", "value": "2C", "onclick": "copy()"},
        {"name": "Cut", "value": "2T", "onclick": "cut()"}
    ]
}
]
}
]
}JSON without root element[
{
    "id": "1",
    "name": "File",
    "value": "F",
    "popup": {
        "menuitem": [
            {"name": "New", "value": "1N", "onclick": "newDoc()"},
            {"name": "Open", "value": "1O", "onclick": "openDoc()"},
            {"name": "Close", "value": "1C", "onclick": "closeDoc()"}
        ]
    },
    {
        "id": "2",
        "name": "Edit",
        "value": "E",
        "popup": {
            "menuitem": [
                {"name": "Undo", "value": "2U", "onclick": "undo()"},
                {"name": "Copy", "value": "2C", "onclick": "copy()"},
                {"name": "Cut", "value": "2T", "onclick": "cut()"}
            ]
        }
    }
}
]

```

Provide META sample

The idea of JSON is flexibility so you don't have to restrict your data feed within few columns.

But at the same time, if you are providing large data set with nested levels, the consumer will go crazy. Provide them with the meta / sample, so it helps them to understand what data to look for and what to skip.

```

{
  "menu": [
    {
      "id": "1",
      "name": "File",
      "value": "F",
      "popup": {
        "menuitem": [
          {"name": "New", "value": "1N", "onclick": "newDoc()"},
          {"name": "Open", "value": "1O", "onclick": "openDoc()"},
          {"name": "Close", "value": "1C", "onclick": "closeDoc()"}
        ]
      }
    }
  ]
}

```

```

        ]
      }
    },
    {
      "id": "2",
      "name": "Edit",
      "value": "E",
      "popup": {
        "menuitem": [
          { "name": "Undo", "value": "2U", "onclick": "undo()" },
          { "name": "Copy", "value": "2C", "onclick": "copy()" },
          { "name": "Cut", "value": "2T", "onclick": "cut()" }
        ]
      }
    }
  ]
}

```

Meta sample can be

menu.id : integer — unique identifier
 menu.name : string — name of the menu
 menu.value : string — interval id of the menu
 menu.popup.menuitem.name : string — name of the submenu
 menu.popup.menuitem.value : string — interval id of the submenu
 menu.popup.menuitem.onclick : string — client event of the submenu

Validating JSON output

Using command line tools like ajv-cli / jsonlint (can be installed via any package manager) will eliminate trouble for consumers.

4. Request - Response Best Practices

HTTP Request is the first step to initiate web request/response communication. Every request is a combination of request header, body and request URL.

HTTP Request Method:

GET() : The get method is used to retrieve information from the given server using a given URL. Requests using GET should only retrieve data and should have no other effect on the data

POST() : A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML form

PUT() : Replaces all current representations of the target resource with the uploaded content

DELETE() : Remove all current representations of the target resource given by a URL

Http Request Throttling:

Throttle Request refers to a process in which a user is allowed to hit the application maximum time in per second or per minute. Throttling is also known as request rate limiting.

- Essential component of Internet security, as DoS attacks can tank a server with unlimited requests.
- Rate limiting also helps make your API scalable by avoid unexpected spikes in traffic, causing severe lag time.

HTTP Response status messages

200 ok : The request succeeded. The result meaning of "success" depends on the HTTP method

201 Created : The request succeeded, and a new resource was created as a result. This is typically the response sent after POST requests, or some PUT requests.

202 Accepted : The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

[203 Non-Authoritative Information](#)

This response code means the returned metadata is not exactly the same as is available from the origin server, but is collected from a local or a third-party copy. This is mostly used for mirrors or backups of another resource. Except for that specific case, the 200 OK response is preferred to this status.

[204 No Content](#)

There is no content to send for this request, but the headers may be useful. The user agent may update its cached headers for this resource with the new ones.

[205 Reset Content](#)

Tells the user agent to reset the document which sent this request.

[206 Partial Content](#)

This response code is used when the [Range](#) header is sent from the client to request only part of a resource.

[207 Multi-Status \(WebDAV\)](#)

Conveys information about multiple resources, for situations where multiple status codes might be appropriate.

208 Already Reported ([WebDAV](#))

Used inside a `<dav:propstat>` response element to avoid repeatedly enumerating the internal members of multiple bindings to the same collection.

226 IM Used ([HTTP Delta encoding](#))

The server has fulfilled a `GET` request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance.

300 Multiple Choices

The request has more than one possible response. The user agent or user should choose one of them. (There is no standardized way of choosing one of the responses, but HTML links to the possibilities are recommended so the user can pick.)

301 Moved Permanently

The URL of the requested resource has been changed permanently. The new URL is given in the response.

302 Found

This response code means that the URI of requested resource has been changed *temporarily*. Further changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.

303 See Other

The server sent this response to direct the client to get the requested resource at another URI with a `GET` request.

304 Not Modified

This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same cached version of the response.

305 Use Proxy Deprecated

Defined in a previous version of the HTTP specification to indicate that a requested response must be accessed by a proxy. It has been deprecated due to security concerns regarding in-band configuration of a proxy.

306 unused

This response code is no longer used; it is just reserved. It was used in a previous version of the HTTP/1.1 specification.

[307 Temporary Redirect](#)

The server sends this response to direct the client to get the requested resource at another URI with same method that was used in the prior request. This has the same semantics as the 302 Found HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a POST was used in the first request, a POST must be used in the second request.

[308 Permanent Redirect](#)

This means that the resource is now permanently located at another URI, specified by the Location: HTTP Response header. This has the same semantics as the 301 Moved Permanently HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a POST was used in the first request, a POST must be used in the second request.

[400 Bad Request](#)

The server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

[401 Unauthorized](#)

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

[402 Payment Required](#) Experimental

This response code is reserved for future use. The initial aim for creating this code was using it for digital payment systems, however this status code is used very rarely and no standard convention exists.

[403 Forbidden](#)

The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401 Unauthorized, the client's identity is known to the server.

[404 Not Found](#)

The server cannot find the requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 `Forbidden` to hide the existence of a resource from an unauthorized client. This response code is probably the most well known due to its frequent occurrence on the web.

405 Method Not Allowed

The request method is known by the server but is not supported by the target resource. For example, an API may not allow calling `DELETE` to remove a resource.

406 Not Acceptable

This response is sent when the web server, after performing [server-driven content negotiation](#), doesn't find any content that conforms to the criteria given by the user agent.

407 Proxy Authentication Required

This is similar to 401 `Unauthorized` but authentication is needed to be done by a proxy.

408 Request Timeout

This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection. This response is used much more since some browsers, like Chrome, Firefox 27+, or IE9, use HTTP pre-connection mechanisms to speed up surfing. Also note that some servers merely shut down the connection without sending this message.

409 Conflict

This response is sent when a request conflicts with the current state of the server.

410 Gone

This response is sent when the requested content has been permanently deleted from server, with no forwarding address. Clients are expected to remove their caches and links to the resource. The HTTP specification intends this status code to be used for "limited-time, promotional services". APIs should not feel compelled to indicate resources that have been deleted with this status code.

411 Length Required

Server rejected the request because the `Content-Length` header field is not defined and the server requires it.

412 Precondition Failed

The client has indicated preconditions in its headers which the server does not meet.

413 Payload Too Large

Request entity is larger than limits defined by server. The server might close the connection or return an `Retry-After` header field.

414 URI Too Long

The URI requested by the client is longer than the server is willing to interpret.

415 Unsupported Media Type

The media format of the requested data is not supported by the server, so the server is rejecting the request.

416 Range Not Satisfiable

The range specified by the `Range` header field in the request cannot be fulfilled. It's possible that the range is outside the size of the target URI's data.

417 Expectation Failed

This response code means the expectation indicated by the `Expect` request header field cannot be met by the server.

418 I'm a teapot

The server refuses the attempt to brew coffee with a teapot.

421 Misdirected Request

The request was directed at a server that is not able to produce a response. This can be sent by a server that is not configured to produce responses for the combination of scheme and authority that are included in the request URI.

422 Unprocessable Entity ([WebDAV](#))

The request was well-formed but was unable to be followed due to semantic errors.

423 Locked ([WebDAV](#))

The resource that is being accessed is locked.

424 Failed Dependency ([WebDAV](#))

The request failed due to failure of a previous request.

425 Too Early Experimental

Indicates that the server is unwilling to risk processing a request that might be replayed.

426 Upgrade Required

The server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol. The server sends an [Upgrade](#) header in a 426 response to indicate the required protocol(s).

428 Precondition Required

The origin server requires the request to be conditional. This response is intended to prevent the 'lost update' problem, where a client GETs a resource's state, modifies it and PUTs it back to the server, when meanwhile a third party has modified the state on the server, leading to a conflict.

429 Too Many Requests

The user has sent too many requests in a given amount of time ("rate limiting").

431 Request Header Fields Too Large

The server is unwilling to process the request because its header fields are too large. The request may be resubmitted after reducing the size of the request header fields.

451 Unavailable For Legal Reasons

The user agent requested a resource that cannot legally be provided, such as a web page censored by a government.

500 Internal Server Error

The server has encountered a situation it does not know how to handle.

501 Not Implemented

The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are GET and HEAD.

502 Bad Gateway

This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.

503 Service Unavailable

The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded. Note that together with this response, a user-

friendly page explaining the problem should be sent. This response should be used for temporary conditions and the `Retry-After` HTTP header should, if possible, contain the estimated time before the recovery of the service. The webmaster must also take care about the caching-related headers that are sent along with this response, as these temporary condition responses should usually not be cached.

504 Gateway Timeout

This error response is given when the server is acting as a gateway and cannot get a response in time.

505 HTTP Version Not Supported

The HTTP version used in the request is not supported by the server.

506 Variant Also Negotiates

The server has an internal configuration error: the chosen variant resource is configured to engage in transparent content negotiation itself, and is therefore not a proper end point in the negotiation process.

507 Insufficient Storage ([WebDAV](#))

The method could not be performed on the resource because the server is unable to store the representation needed to successfully complete the request.

508 Loop Detected ([WebDAV](#))

The server detected an infinite loop while processing the request.

510 Not Extended

Further extensions to the request are required for the server to fulfill it.

511 Network Authentication Required

Indicates that the client needs to authenticate to gain network access.

Most apps that use a modern web framework will have one or more [REST APIs](#). [REST is a simple and flexible way](#) of structuring a web API. It's not a standard or protocol, but rather a set of architectural constraints.

5. Web Security Practices

Use OAuth2 for single sign on (SSO) with OpenID Connect

Use API keys to give existing users programmatic access

Encourage using good secrets management for API keys

Choose when to enforce authorization with request-level authorization

Leave the rest of the authorization to the app/business logic

In summary: use good libraries

6. Interview Question on Web Back-End Development

1. What is your favorite programming language? And why?
2. Explain what are the key responsibilities of a Web Developer?
3. What web developer should know?
4. Explain what is CORS? How does it work?
5. List out the advantage of HTTP/2 as compared with HTTP 1.1?
6. Explain what is an ETag and how does it work?
7. Explain what is long polling?
8. Explain what is DTD (Document Type Declaration)? Mention what is the difference between CDATA and PCDATA in DTD?
9. Mention some tips you can use to reduce the load time of a web application that you have written?
10. Mention what is the correct way to include JavaScript into your HTML?
11. Explain in CSS, how can you make a form element's background color change when the user is entering text? Does this work in all browsers?
12. Explain what is the difference between cookies and local storage?
13. What do you understand by NoSQL databases? Explain.
14. What is the difference between software architecture and software design?
15. What is JavaScript, and why is it used?
16. What is your approach to debugging?
17. How would you find the most expensive queries in an application?
18. What is the difference between an acceptance test and a functional test?
19. Which sorting algorithm should you use and when?
20. What is the meaning of "high cohesion" and "loose coupling"?

21. Why should you use microservices architecture?
22. Explain the difference between cohesion and coupling?
23. When is refactoring useful?
24. What is your experience with object-oriented programming (OOP)?
25. Tell us about a time when you received criticism about your work and how you handled it.
26. If you have to deliver negative feedback to members of your development team, how would you do it?
27. Tell me about your favorite development project to date and what it was like from start to finish.
28. What is your experience with GoTo, and do you prefer structured programming?
29. Tell me about the largest web application you have ever worked on? What coding were you responsible for?
30. What's your preferred type of development environment?
31. Which programming languages do you want to work with and why?
32. What is Event Loop?
33. Explain the difference between REST and GraphQL
34. What is Long Polling?
35. What is Pair-Programming?
36. Explain Prototype Inheritance in Javascript?
37. What are the Advantages and Disadvantages of Using "use strict"?
38. What do you think of AMD vs CommonJS?
39. What is Continuous Integration?
40. What is Multi-Threading?
41. What is CORS?