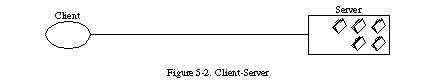
What is REST

REST is an acronym for REpresentational State Transfer. REST is an architectural style for distributed hypermedia systems. It’s a set of rules which help programs to communicate with each other. The developers create APIs on the servers, which permit clients to talk to them.

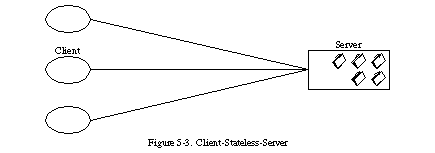
### ***1) Client – Server Architecture style***

The first constraints added to our hybrid style are those of the client-server architectural style ([Figure 5-2](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_2)), described in [Section 3.4.1](https://www.ics.uci.edu/~fielding/pubs/dissertation/net_arch_styles.htm#sec_3_4_1). Separation of concerns is the principle behind the client-server constraints. By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components. Perhaps most significant to the Web, however, is that the separation allows the components to evolve independently, thus supporting the Internet-scale requirement of multiple organizational domains.



### ***2) Stateless***

We next add a constraint to the client-server interaction: communication must be stateless in nature, as in the client-stateless-server (CSS) style of [Section 3.4.3](https://www.ics.uci.edu/~fielding/pubs/dissertation/net_arch_styles.htm#sec_3_4_3) ([Figure 5-3](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_3)), such that each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

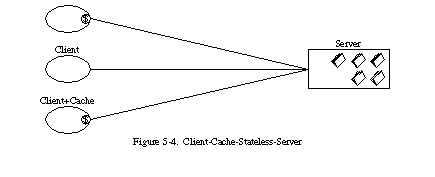


This constraint induces the properties of visibility, reliability, and scalability. Visibility is improved because a monitoring system does not have to look beyond a single request datum in order to determine the full nature of the request. Reliability is improved because it eases the task of recovering from partial failures [[133](https://www.ics.uci.edu/~fielding/pubs/dissertation/references.htm#ref_133)]. Scalability is improved because not having to store state between requests allows the server component to quickly free resources, and further simplifies implementation because the server doesn't have to manage resource usage across requests.

Like most architectural choices, the stateless constraint reflects a design trade-off. The disadvantage is that it may decrease network performance by increasing the repetitive data (per-interaction overhead) sent in a series of requests, since that data cannot be left on the server in a shared context. In addition, placing the application state on the client-side reduces the server's control over consistent application behavior, since the application becomes dependent on the correct implementation of semantics across multiple client versions.

### ***3) Cache***

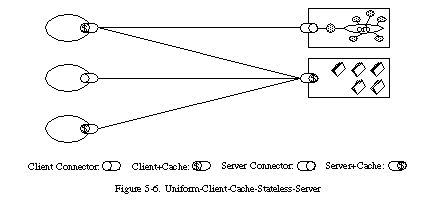
In order to improve network efficiency, we add cache constraints to form the client-cache-stateless-server style of [Section 3.4.4](https://www.ics.uci.edu/~fielding/pubs/dissertation/net_arch_styles.htm#sec_3_4_4) ([Figure 5-4](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_4)). Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.



The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions. The trade-off, however, is that a cache can decrease reliability if stale data within the cache differs significantly from the data that would have been obtained had the request been sent directly to the server.

### ***4) Uniform Interface***

* *The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components (*[*Figure 5-6*](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_6)*). By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. Implementations are decoupled from the services they provide, which encourages independent evolvability. The trade-off, though, is that a uniform interface degrades efficiency, since information is transferred in a standardized form rather than one which is specific to an application's needs. The REST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction.*

**

* *In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components.*

### ***5) Layered System***

This is the fifth constraint in the *REST* architectural style. According to this constraint the system implementation should be layered. Each layer abstracting out certain functionality of the overall system. A layer should not know about the existence of other layers apart from the layers that it directly interacts with.

### ***6) Code on demand***

This is an optional constraint that *RESTful* architecture imposes. According to this constraint a client can extend its functionality by downloading code from the Server This code can come in the form of Applets or Scripts. Javascript is one example of how code on demand can help clients extend their functionalities.

With the ability of a Server sending code to the Client, comes the security concerns.

### HTTP Methods for REST API Automation Testing

REST API uses five HTTP methods to request a command:

**GET**: To retrieve the information at a particular URL.

**PUT**: To update the previous resource or create new information at a particular URL.

**PATCH**: For partial updates.

**POST**: It is used to develop a new entity. Moreover, it is also used to send information to servers, such as uploading a file, customer information, etc.

**DELETE**: To delete all current representations at a specific URL.

### HTTP Status Codes

Status codes are the response given by a server to a client’s request. They are classified into five categories:

1. 1xx (100 – 199): The response is informational
2. 2xx (200 – 299): Assures successful response
3. 3xx (300 – 399): You are required to take further action to fulfill the request
4. 4xx (400 – 499): There’s a bad syntax and the request cannot be completed
5. 5xx (500 – 599): The server entirely fails to complete the request

### Testing GET Request

**Note:** After clicking this link (<https://reqres.in/api/users?page=2>) you will see the JSON code.

But the best thing about Postman is that if you have the API URL, you can simply paste it to get the status code, and the same rule applies to all other methods.

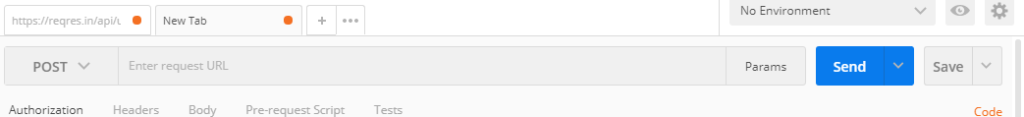
**Test Tutorial:** After launching Postman, select the GET method and copy the GET request link (<https://reqres.in/api/users?page=2>) and paste it in the ‘Enter request URL’ field. Once done, click ‘Send’ to see the ‘Status.’

**Results:** You can see that the status is ‘200 OK.’

**Interpretation:** ‘OK’ signifies that the GET request is functioning without errors as the status code is exactly 200.

### Testing POST Request

**Step 1:** Click on ‘New Tab’ and select ‘POST’ from the drop-down menu.



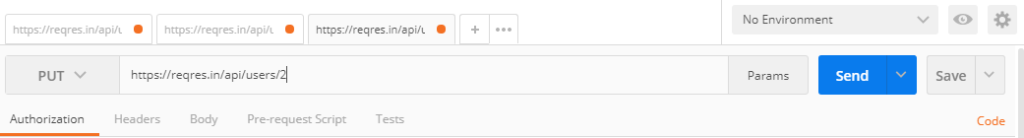
**Step 2**: Copy and paste the POST request URL (<https://reqres.in/api/users>) in the ‘Enter Request URL’ field and hit the ‘Send’ button.

**Results**: The status code is ‘200 Created.’

**Interpretation**: ‘Created’ means that the POST request has succeeded and a new resource has been generated.

### Testing PUT Request

**Step 1:** Open ‘New Tab’ and choose ‘PUT’ from the options.



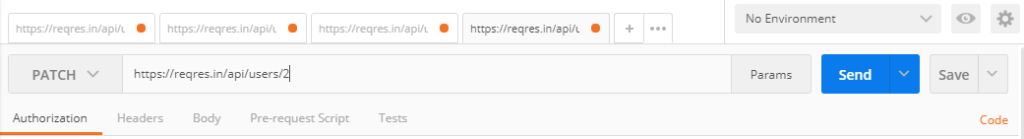
**Step 2:** Copy and paste the PUT request URL (<https://reqres.in/api/users/2>) in the ‘Enter Request URL’ field, and click ‘Send.’

**Results**: Status is ‘200 OK.’

**Interpretation**: PUT request is functioning.

### Testing PATCH Request

**Step 1**: Select ‘PATCH’ in the ‘New Tab.’



**Step 2**: Paste the PATCH request URL in the field and click ‘Send.’

**Results**: Status code is ‘200 OK.’

**Interpretation**: PATCH request is functioning properly.

### Testing DELETE Request

**Step 1**: Create a ‘New Tab’ and this time select ‘DELETE’ from the menu.



**Step 2**: Enter the DELETE request URL in the field and click the ‘Send’ button again.

**Result**: You can see that the Status is ‘204 No Content.’

**Interpretation**: As the link has no content to be deleted, therefore, we get a ‘204’ response. However, if there were any content on the URL, it would surely be deleted with this command as the status code is in the typical range.

HTTP Methods

REST APIs enable you to develop any kind of web application having all possible CRUD (create, retrieve, update, delete) operations.

[REST guidelines](https://restfulapi.net/rest-architectural-constraints/) suggest using a specific HTTP method on a particular type of call made to the server (though technically it is possible to violate this guideline, yet it is highly discouraged).

## **1. HTTP GET**

Use GET requests to retrieve resource representation/information only – and not to modify it in any way. As GET requests do not change the state of the resource, these are said to be safe methods.

Additionally, GET APIs should be idempotent, which means that making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server.

For any given HTTP GET API, if the resource is found on the server, then it must return HTTP response code 200 (OK) – along with the response body, which is usually either XML or JSON content (due to their platform-independent nature).

In case the resource is NOT found on the server then it must return HTTP response code 404 (NOT FOUND).

Similarly, if it is determined that the GET request itself is not correctly formed then the server will return the HTTP response code 400 (BAD REQUEST).

## What are API Query Parameters?

API Query parameters can be defined as the optional key-value pairs that appear after the question mark in the URL. Basically, they are extensions of the URL that are utilized to help determine specific content or action based on the data being delivered. Query parameters are appended to the end of the URL, using a ‘?’. The question mark sign is used to separate path and query parameters.

If you want to add multiple query parameters, an ‘&’ sign is placed in between them to form what is known as a query string. It can feature various object types with distinct lengths such as arrays, strings, and numbers.

## Query Parameter Examples

1. https://example.com/articles?sort=ASC&page=2

In this URL, there are two query parameters, sort, and page, with ASC and 2 being their values, respectively.

2. http//www.techopedia.com/search.aspx?q=database&ion-all

In the URL above, the bolded values after the ‘?’ are the query parameters, q=database&ion-all (query string).

## **6. Summary of HTTP Methods**

The below table summarises the use of HTTP methods discussed above.

| **HTTP Method** | **CRUD** | **Collection Resource (e.g. /users)** | **Single Resouce (e.g. /users/123)** |
| --- | --- | --- | --- |
| POST | Create | 201 (Created), ‘Location’ header with link to /users/{id} containing new ID | Avoid using POST on a single resource |
| GET | Read | 200 (OK), list of users. Use pagination, sorting, and filtering to navigate big lists | 200 (OK), single user. 404 (Not Found), if ID not found or invalid |
| PUT | Update/Replace | 405 (Method not allowed), unless you want to update every resource in the entire collection of resource | 200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid |
| PATCH | Partial Update/Modify | 405 (Method not allowed), unless you want to modify the collection itself | 200 (OK) or 204 (No Content). Use 404 (Not Found), if ID is not found or invalid |
| DELETE | Delete | 405 (Method not allowed), unless you want to delete the whole collection — use with caution | 200 (OK). 404 (Not Found), if ID not found or invalid |

# REST API - Response Codes and Statuses

| **Code** | **Status** | **Description** |
| --- | --- | --- |
| 200 | OK | The request was successfully completed. |
| 201 | Created | A new resource was successfully created. |
| 400 | Bad Request | The request was invalid. |
| 401 | Unauthorized | The request did not include an authentication token or the authentication token expired. |
| 403 | Forbidden | The client did not have permission to access the requested resource. |
| 404 | Not Found | The requested resource was not found. |
| 405 | Method Not Allowed | The HTTP method in the request was not supported by the resource. For example, the DELETE method cannot be used with the Agent API. |
| 409 | Conflict | The request could not be completed due to a conflict. For example, POST ContentStore Folder API cannot complete if the given file or folder name already exists in the parent location. |
| 500 | Internal Server Error | The request was not completed due to an internal error on the server side. |
| 503 | Service Unavailable | The server was unavailable. |