**Spring REST**

**Understanding REST**

REST (Representational State Transfer) was introduced and defined in 2000 by Roy Fielding in his [doctoral dissertation](https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm). REST is an architectural style for designing distributed systems. It is not a standard but a set of constraints, such as being stateless, having a client/server relationship, and a uniform interface. REST is not strictly related to HTTP, but it is most commonly associated with it. By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running.

## **Principles of REST**

### Client–server architecture

The principle behind the client–server constraints is the separation of concerns. Separating the user interface concerns from the data storage concerns improves the portability of the user interface across multiple platforms. It also improves 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.

### Statelessness

The client–server communication is constrained by no client context being stored on the server between requests. Each request from any client contains all the information necessary to service the request, and session state is held in the client. The session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be *in transition*. The representation of each application state contains links that can be used the next time the client chooses to initiate a new state-transition.

### Cacheability

### As on the World Wide Web, clients and intermediaries can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable or not to prevent clients from getting stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

### Layered system

### A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers can improve system scalability by enabling load balancing and by providing shared caches. They can also enforce security policies.

### Code on demand

Servers can temporarily extend or customize the functionality of a client by transferring executable code: for example, compiled components such as [Java applets](https://en.m.wikipedia.org/wiki/Java_applet), or client-side scripts such as [JavaScript](https://en.m.wikipedia.org/wiki/JavaScript).

### Uniform interface

The uniform interface constraint is fundamental to the design of any RESTful system. It simplifies and decouples the architecture, which enables each part to evolve independently. The four constraints for this uniform interface are:

Resource identification in requests

Individual resources are identified in requests, for example using [URIs](https://en.m.wikipedia.org/wiki/Uniform_resource_identifier) in RESTful Web services. The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server could send data from its database as [HTML](https://en.m.wikipedia.org/wiki/HTML), [XML](https://en.m.wikipedia.org/wiki/XML) or as [JSON](https://en.m.wikipedia.org/wiki/JSON)—none of which are the server's internal representation.

Resource manipulation through representations

When a client holds a representation of a resource, including any [metadata](https://en.m.wikipedia.org/wiki/Metadata) attached, it has enough information to modify or delete the resource.

Self-descriptive messages

Each message includes enough information to describe how to process the message. For example, which parser to invoke can be specified by a [media type](https://en.m.wikipedia.org/wiki/Media_type).

Hypermedia as the engine of application state ([HATEOAS](https://en.m.wikipedia.org/wiki/HATEOAS))

Having accessed an initial URI for the REST application—analogous to a human Web user accessing the [home page](https://en.m.wikipedia.org/wiki/Home_page) of a website—a REST client should then be able to use server-provided links dynamically to discover all the available actions and resources it needs. As access proceeds, the server responds with text that includes [hyperlinks](https://en.m.wikipedia.org/wiki/Hyperlink) to other actions that are currently available. There is no need for the client to be hard-coded with information regarding the structure or dynamics of the application.

## **HTTP methods**

Use HTTP methods to map CRUD (create, retrieve, update, delete) operations to HTTP requests.

### GET

Retrieve information. GET requests must be safe and [idempotent](https://en.wikipedia.org/wiki/Idempotence#Computer_science_meaning), meaning regardless of how many times it repeats with the same parameters, the results are the same. They can have side effects, but the user doesn't expect them, so they cannot be critical to the operation of the system. Requests can also be partial or conditional.

### POST

Request that the resource at the URI do something with the provided entity. Often POST is used to create a new entity, but it can also be used to update an entity.

### PUT

Store an entity at a URI. PUT can create a new entity or update an existing one. A PUT request is idempotent. Idempotency is the main difference between the expectations of PUT versus a POST request.

### PATCH

Update only the specified fields of an entity at a URI. A PATCH request is neither safe nor idempotent (RFC 5789). That's because a PATCH operation cannot ensure the entire resource has been updated.

### DELETE

Request that a resource be removed; however, the resource does not have to be removed immediately. It could be an asynchronous or long-running request.

The Status-Code element in a server response, is a 3-digit integer where the first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

|  |  |
| --- | --- |
| **S.N.** | **Code and Description** |
| 1 | **1xx: Informational**  It means the request has been received and the process is continuing. |
| 2 | **2xx: Success**  It means the action was successfully received, understood, and accepted. |
| 3 | **3xx: Redirection**  It means further action must be taken in order to complete the request. |
| 4 | **4xx: Client Error**  It means the request contains incorrect syntax or cannot be fulfilled. |
| 5 | **5xx: Server Error**  It means the server failed to fulfill an apparently valid request. |

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all the registered status codes. Given below is a list of all the status codes.

## **1xx: Information**

|  |  |
| --- | --- |
| **Message** | **Description** |
| 100 Continue | Only a part of the request has been received by the server, but as long as it has not been rejected, the client should continue with the request. |
| 101 Switching Protocols | The server switches protocol. |

## **2xx: Successful**

|  |  |
| --- | --- |
| **Message** | **Description** |
| 200 OK | The request is OK. |
| 201 Created | The request is complete, and a new resource is created . |
| 202 Accepted | The request is accepted for processing, but the processing is not complete. |
| 203 Non-authoritative Information | The information in the entity header is from a local or third-party copy, not from the original server. |
| 204 No Content | A status code and a header are given in the response, but there is no entity-body in the reply. |
| 205 Reset Content | The browser should clear the form used for this transaction for additional input. |
| 206 Partial Content | The server is returning partial data of the size requested. Used in response to a request specifying a *Range* header. The server must specify the range included in the response with the *Content-Range* header. |

## **3xx: Redirection**

|  |  |
| --- | --- |
| **Message** | **Description** |
| 300 Multiple Choices | A link list. The user can select a link and go to that location. Maximum five addresses. |
| 301 Moved Permanently | The requested page has moved to a new url. |
| 302 Found | The requested page has moved temporarily to a new url. |
| 303 See Other | The requested page can be found under a different url. |
| 304 Not Modified | This is the response code to an *If-Modified-Since* or *If-None-Match* header, where the URL has not been modified since the specified date. |
| 305 Use Proxy | The requested URL must be accessed through the proxy mentioned in the *Location* header. |
| 306 *Unused* | This code was used in a previous version. It is no longer used, but the code is reserved. |
| 307 Temporary Redirect | The requested page has moved temporarily to a new url. |

## **4xx: Client Error**

|  |  |
| --- | --- |
| **Message** | **Description** |
| 400 Bad Request | The server did not understand the request. |
| 401 Unauthorized | The requested page needs a username and a password. |
| 402 Payment Required | *You cannot use this code yet*. |
| 403 Forbidden | Access is forbidden to the requested page. |
| 404 Not Found | The server cannot find the requested page. |
| 405 Method Not Allowed | The method specified in the request is not allowed. |
| 406 Not Acceptable | The server can only generate a response that is not accepted by the client. |
| 407 Proxy Authentication Required | You must authenticate with a proxy server before this request can be served. |
| 408 Request Timeout | The request took longer than the server was prepared to wait. |
| 409 Conflict | The request could not be completed because of a conflict. |
| 410 Gone | The requested page is no longer available. |
| 411 Length Required | The "Content-Length" is not defined. The server will not accept the request without it. |
| 412 Precondition Failed | The pre-condition given in the request evaluated to false by the server. |
| 413 Request Entity Too Large | The server will not accept the request, because the request entity is too large. |
| 414 Request-url Too Long | The server will not accept the request, because the url is too long. Occurs when you convert a "post" request to a "get" request with a long query information. |
| 415 Unsupported Media Type | The server will not accept the request, because the media type is not supported. |
| 416 Requested Range Not Satisfiable | The requested byte range is not available and is out of bounds. |
| 417 Expectation Failed | The expectation given in an Expect request-header field could not be met by this server. |

## **5xx: Server Error**

|  |  |
| --- | --- |
| **Message** | **Description** |
| 500 Internal Server Error | The request was not completed. The server met an unexpected condition. |
| 501 Not Implemented | The request was not completed. The server did not support the functionality required. |
| 502 Bad Gateway | The request was not completed. The server received an invalid response from the upstream server. |
| 503 Service Unavailable | The request was not completed. The server is temporarily overloading or down. |
| 504 Gateway Timeout | The gateway has timed out. |
| 505 HTTP Version Not Supported | The server does not support the "http protocol" version. |