HTTP Methods and REST APIs

Estimated time needed: **30** minutes

Objectives

After reading this document, you should be able to:

- Define terms related to HTTP methods
- Explain guidelines and best practices for writing REST APIs

The internet relies on a client-server architecture. The end-user interfaces with the client, while the servers house the **services**[1] that operate the applications, the business logic, and the data. Clients communicate with the servers to achieve desired functionality for the end user. Data is transferred between client and server using hypertext transfer protocol, more commonly known as **HTTP**[2]. This communication usually happens via **APIs**[3]. In 2000, a set of guidelines for writing these APIs for a client-server architecture was developed called **REST**[4] APIs.

The acronym "REST" stands for REpresentational State Transfer. Before explaining this term in more detail, let's discuss HTTP methods and some terminology first.

In a client/server architecture, the applications are composed of one or more services that reside on the servers. These services contain **resources**[5]. The client makes a **request**[6] for a resource via a **request object**[7] using a **route**[8] that has an **endpoint**[9] within the service. The application sends a **response object**[10] back in **response**[11] to the client to honor that request.

A request object contains three parts, a **URL**[12], a **request header**[13], and a **request body**[14]. The server uses the URL to identify the service and the endpoint within the service being acted upon. The URL contains four parts: a **protocol**[15], a **hostname**[16], a **path**[17], and a **query string**[18]. The request header contains metadata about the resource of the requesting client, such as the user **agent**[19], **host**[20], **content type**[21], **content length**[22], and what type of data the client should expect in the response.

The server responds with a response object consisting of a **header**[23], a **body**[24], and a **status code**[25]. The response object body often contains a **JSON**[26] **payload**[27] to provide the data back to the client.

There are a number of HTTP methods that can be used in the REST API that allow interaction between the client and a service. The most common methods are **GET**[28],**POST**[29], **PUT**[30], **DELETE**[31], and **PATCH**[32].

The name of the method describes what happens to the resource when the method is applied. PUT and DELETE methods result in **idempotent**[33] data if the same API method is called multiple times.

HTTP has three ways to pass parameters: the **URL path parameter**[34], the **URL query parameter**[35], and the **header parameter**[36]. The path and query parameters are passed as part of the URL, but the header parameter is passed by the browser directly to the service.

When the service completes a request, it returns a response. An HTTP status code should be part of that response. The HTTP status code indicates whether the response has been completed or not. Response code categories are shown in the following table.

Status Code Range	Meaning
200-299	Everything is OK
300-399	Resource has moved
400-499	Client-side error
500-599	Server-side error

As mentioned earlier, REST is a set of guidelines. There are five requirements for an API to be considered RESTful, plus one optional criterion:

1. The API leverages a client-server architecture made up of resources that are managed and delivered via HTTP.
2. Communication between client and server is **stateless**[37].
3. Data is **cacheable**[38] to improve performance on the client side.
4. The interface is transferred in a standard format such that the requested resources stored on the server are separated from the representation sent to the client. The representation sent to the client contains sufficient data so that the client can manipulate the representation.
5. Requests and responses communicate through different layers, such as **middleware**[39]. The client and server often do not communicate directly with each other.

6. (Optional) Resources are usually static but can also contain executable code. The code should only be executed when the client requests it.

When the client makes a request, it also must pass information about its state to the server. Every communication between the client and server should contain all the information needed to perform the request. The client, not the service, maintains the state. Each request must contain the requisite information so the server understands the request. So, for example, if the user is viewing a database record and that record needs to be updated, the client must also send which record needs updating. The server doesn't know which record is currently being viewed.

An `@app.route()` method is used when defining a RESTful service. This method takes two parameters: the route from the URL to the service being acted upon and an optional HTTP method parameter such as POST, GET, etc. The route parameter can include variables, such as `<username>`. The root of a route is represented by '/'. For example, if you want the route to be www.mywebsite.com/accounts, you only need to specify '/accounts' as the route parameter in the `@app.route()` function.

REST APIs have the following characteristics:

- Resource-based; that is, they describe sets of resources
- Only contain nouns, not verbs
- Use singular nouns when referring to a singular resource or plural nouns when referring to a collection of resources
- Always identified by URLs

NOT RESTful APIs	**RESTful equivalents**
GET http://api.myapp.com/getUser/123	GET http://api.myapp.com/users/123
POST http://api.myapp.com/addUser	POST http://api.myapp.com/users
GET http://api.myapp.com/removeUser/123	DELETE http://api.myapp.com/users/123

URL format guidelines

- Should use a slash '/' to denote a hierarchical relationship in the directory structure
- Should avoid using a trailing slash, e.g., /resource/
- Should use hyphens, not camel case, e.g., /my-resource, not /myResource
- Should not use an underscore '_' in the URL, e.g., /my-resource, not /my_resource
- Should use lowercase
- Should not use a period '.' in a URL
- May contain multiple subordinate resources and IDs in the URL, e.g., GET /resource/{id}/subordinate/{id}

Terms and Definitions

Term	**Description**
1.Service	A software component that makes up part of an application and serves a specific purpose. Generally, a service takes input from a client or another service and produces an output.
2.HTTP	The protocol used by a client-server architecture on the internet for fetching or exchanging resources data.
3.API	An Application Programming Interface is a set of definitions and protocols that allow two services to communicate with each other. The API requests data that can be exchanged between a resource and the results that are returned from that resource.
4.REST	A set of architectural guidelines that describe how to write an interface (API) between two components, usually a client and server, that describe how these components communicate with each other. REST stands for REpresentational State Transfer. REST describes a standard way to identify and manipulate resources. REST ensures the messages passed between the client and server are self-descriptive and define how the client interacts with the server to access resources on the server.
5.Resource	A resource is the fundamental concept of a RESTful API. It is an object that has a defined type, associated data, relationships to other resources, and a set of methods that can operate on it. A resource is commonly defined in JSON format but can also be XML.
6.Request	A request is made by a client to a host on a server to access a resource. The client uses parts of a URL to determine the information needed from the resource. Most common request methods include GET, POST, PUT, PATCH, and DELETE but also include HEAD, CONNECT, OPTIONS, TRACE, and PATCH.
7.Request Object	Contains the HTTP request data. It contains three parts: a URL, a header, and a body.

8.Route	The combination of an HTTP method and the path to the resource from the root of the path.
9.Endpoint	The location of the resource specified by a REST API that is being accessed on the server. It is usually identified through the URL in the HTTP method of the API.
10.Response Object	Contains the HTTP response data in response to a request. It contains a header, a body, and a status.
11.Response	A response is made by a server and sent to a client to either provide the client with the requested resource, tell the client the requested action has been completed, or let the client know there has been an error processing the request.
12.URL	A "Uniform Resource Identifier" is used interchangeably with the term URL. They are part of a RESTful API that locates the endpoint of the requested resource and contains the data about how that endpoint should be manipulated. The client issues an HTTP request using the URI/URL to manipulate the resource. They should consist of four parts: the hostname, the path, the header, and a query string.
13.Request Header	Information passed to the server about the retrieved resource or the requesting client. Examples include: • Method with endpoint: POST /car-reviews • User-agent: The type of browser the client is using. • Host: A computer on a network that communicates with other hosts. • ContentType: The media type of a resource such as text, audio, or an image. • Content length: The number of bytes of data being sent in a response. • Accept-Encoding: Expected return data format, e.g., application/json • Connection information
14.Request Body	Provides the data being passed to the server.
15.Protocol	Tells the service how the data is to be transferred between the server and the client.
16.Hostname	The name of a device on a network, also often called the site name.
17.Path	The path identifies the location of the resource in the service and its endpoint. For example: https://www.customerservice/customers/{customer_id}
18.Query String	The part of a URL that contains the query.
19.User-agent	The type of browser the client is using.
20.Host	A computer on a network that communicates with other hosts.
21.Content type	The media type of a resource such as text, audio, or an image.
22.Content length	The number of bytes of data being sent in a response.
23.Response Header	Contains metadata about the response, such as a time stamp, caching control, security info, content type, and the number of bytes in the response body.
24.Response Body	The data from the requested resource is sent back to the client.
25.Response Status	The return code that communicates the result of the request’ s status to the client.
26.JSON	"JavaScript Object Notation" is a format for storing and transporting data, usually as a way to send data from a service on a server to the client. It consists of key-value pairs and is self-describing. The format of JSON data is the same as the code for creating JavaScript objects, making it easy to convert this data into JavaScript objects but can be written in any programming language. JSON has three data types: scalars (numbers, strings, Booleans, null), arrays, and objects.
27.Payload	The payload is the data in the body of a response being transported from a server to the client due to an API request.
28.GET	The GET method is used as a request that retrieves a representation of a resource. GET() should never modify a resource and only return a representation of the requested resource.
29.POST	HTTP method that sends data to the server to create a resource and should return 201_CREATED status code.

30.PUT	HTTP method that updates a resource or replaces an existing one. Calling PUT multiple times in a row does not have side effects, whereas POST does. It should return a 200_OK code if the resource exists and can be updated or return a 404_NOT_FOUND code if the resource doesn't exist.
31.DELETE	HTTP method that deletes a resource and returns 204_NO_CONTENT if the resource exists and can be deleted by the server or if the resource cannot be found, which means it has already been deleted.
32.PATCH	HTTP method that applies partial modifications to a resource.
33.Idempotent	Describes an element of a set that remains unchanged when making multiple identical requests. PUT and DELETE methods result in idempotent data if the same API method is called multiple times.
34.URL Path Parameter	Passed into the operation by the client as a variable in the URL's path.
35.URL Query Parameter	Contains key-value pairs, usually in JSON format, and are separated from the path by a '?'. If there are multiple key-value pairs, they should be separated by an '&'. The query can be used to pass in a filter to be applied to the results that are returned by the operation.
36.Header Parameter	Contains additional metadata about the query, such as identifying the client that is calling the operation.
37.Stateless	All requests from a client to a server for resources happen in isolation from each other. The server is unaware of the application's state on the client, so this information needs to be passed with every request.
38.Cacheable	The ability to store data on the client so that data can be used in a future request.
39.Middleware	Software that sits between applications, databases, or services and allows those different technologies to communicate.