HTTP: Differences Between PUT and PATCH

Last modified: November 6, 2022

by Vinicius Fulber-Garcia

Networking Security

HTTP

If you have a few years of experience in Computer Science or research, and you're interested in sharing that experience with the community, have a look at our **Contribution Guidelines**.

1. Introduction

The Hypertext Transfer Protocol (HTTP) is a widely employed

communication protocol. This protocol has been used on the Internet since the 90's to implement distributed hypermedia information systems.

To do that, HTTP provides several standard methods. These methods have properties and objectives that the developers of HTTP systems must attempt to. However, some HTTP methods have very similar characteristics. Thus, we should carefully analyze these similar HTTP methods before programming them in an HTTP system. For instance, the methods of PUT and PATCH have comparable features that can get us confused. But, as we'll see in this article, they are not the same.

In this tutorial, we'll explore the PUT and PATCH HTTP methods. First, we'll have a review of HTTP. In this review, we'll pay special attention to the available HTTP methods and their properties. Thus, we'll in-depth study the PUT and PATCH methods, getting a detailed explanation about them and some usage examples. At last, we compare these methods in a systematic summary.

2. Outlining HTTP

HTTP is an application layer protocol that enables clients and servers to communicate and exchange data. Operationally, HTTP works over the Internet Protocol (IP) at the network layer and the Transmission Control Protocol (TCP) at the transport layer. Furthermore, it adopts a request/response model. Thus, clients send requests and wait for server responses.

In short, an HTTP request contains a Uniform Resource Identifier (URI) and the desired method. Furthermore, it is possible to include, if necessary, request modifiers and body content. Server responses, in turn, contain a success or error code and the payload, which is called entity.

2.1. HTTP Methods Summary

Requesting methods in HTTP indicate the action executed over a particular resource. There exist nine methods, eight of them specified in the context of the base document of HTTP 1.1 (RFC 7231), and another one (PATCH) specified in a particular document (RFC 5789):

- **OPTIONS**: describes the supported HTTP methods of resources. Furthermore, it informs these resources options, requirements, and parameters
- **GET**: employed for receiving information about a resource. In this way, this method can both return already available data or trigger a data-producing process in the server
- **HEAD**: returns only the metainformation of HTTP headers of a GET method. It means that the body content of an entity isn't provided
- **POST**: designed to send a new entity of a resource within the request. Thus, the server subordinates the received entity to the resource
- **PUT**: sends an enclosed entity of a resource to the server. If the entity already exists, the server updates its data. Otherwise, the server creates a new entity
- **DELETE**: triggers the deletion of an entity of a resource. The request must inform the target entity
- **TRACE**: a method with debugging purposes. It returns the entire request to the client. Typically, gateways and proxies tests use this method
- **CONNECT**: employed for tunneling communications. For example, it is useful to establish connections with SSL-enabled websites
- **PATCH**: allows the modification of an entity of a resource. So, it can be applied to change only particular portions of an entity data

2.2. HTTP Methods Properties

There are two relevant properties of HTTP methods: safety and idempotency:

- **Safety**: a method is safe if it doesn't intend to execute an action than an information retrieval when requested. We can perceive safety as the

absence of side effects on the server due to the processing of a method. However, if side effects occur anyway, they are just a consequence, not a request of the users. By convention, OPTIONS, HEAD, and GET are safe methods

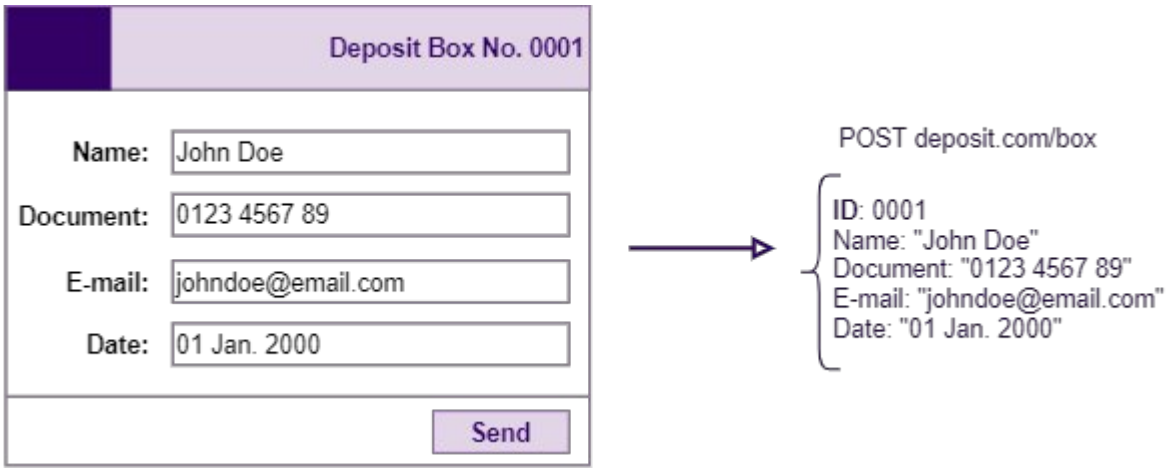
- **Idempotency**: this property means that the side-effects to the server on processing the same request multiple times are equivalent to processing the request a single time. Thus, GET, HEAD, PUT, DELETE, OPTIONS, and TRACE are idempotent methods

3. PUT Method

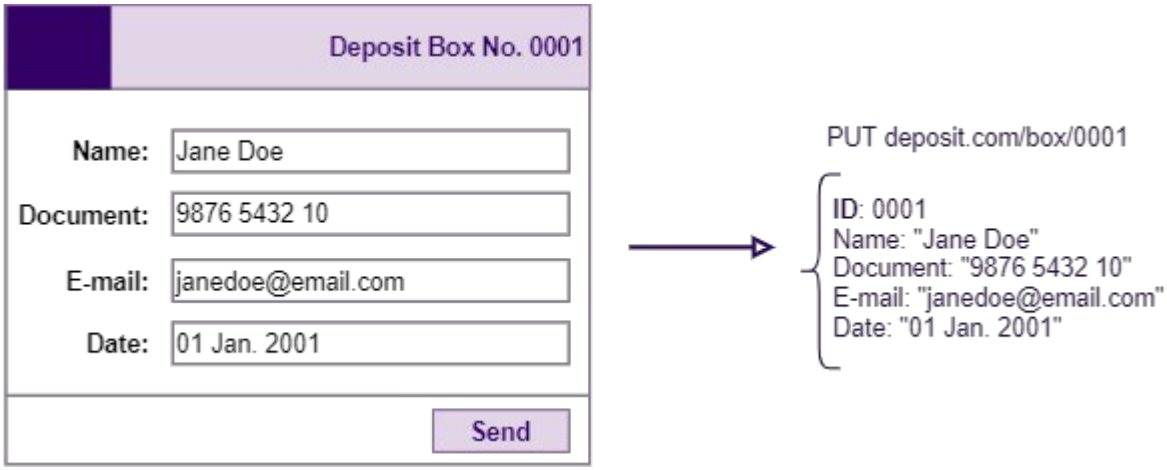
Clients use the PUT method to set up an entity of a resource into an HTTP server. This setup process, in turn, can occur in two forms:

- The entity doesn't exist, and the server creates a new entity for the requested resource and responds with a success code 201 to the client
- The entity already exists; the server updates the entity and responds with a success code 200 or 204 to the client. Furthermore, if an error happens while processing a PUT request, the server should respond to the client with the proper error code, typically a 4xx or 5xx

Let's consider a simple example of registering the lessor of a deposit box. So, an agent collects information about the lessor and POST the first registration of a particular deposit box:



After some time, the deposit box is transferred to another lessor. Thus, the agent collects the information of the new lessor and updates all the data of the deposit box through a PUT method:



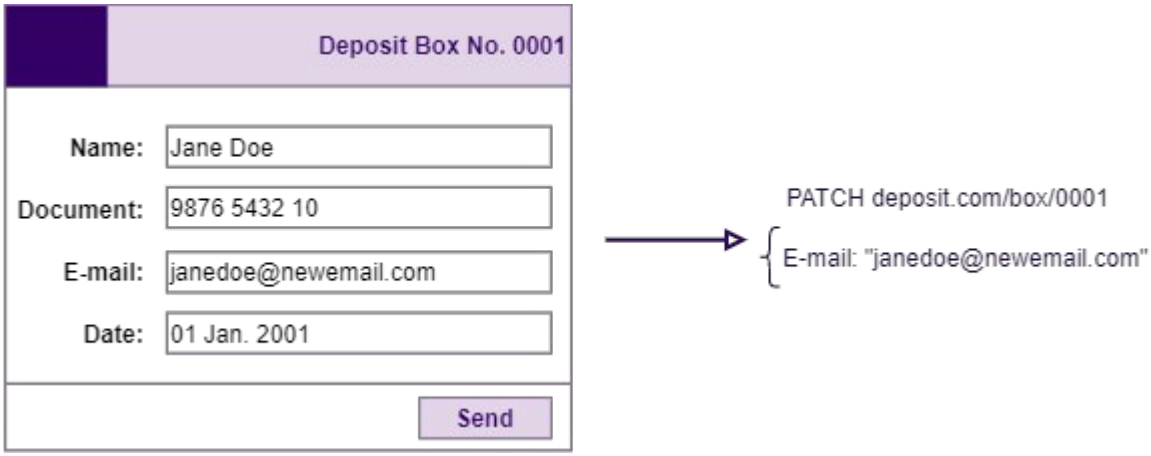
It is relevant to observe that the PUT method sets up the entity with the exact information provided in the request. In this way, the request must contain the entire entity, not only specific fields. But, once the deposit box was transferred to a new lessor, it is expected that all the personal information (or most of it, at least) will change. So, PUT fits well in this scenario.

4. PATCH Method

The PATCH method applies partial modifications to entities of a resource. The PATCH method executes the requested changes atomically. It means that if the server can't satisfy all the requested changes, it doesn't modify the target entity. In such a way, if the request is successfully executed, the server returns the success code 204 to the client. Otherwise, the server returns an error code.

To see a concrete scenario of the PATCH method employment, let's consider a new case of the example presented in the previous section. After some time, the new lessor of the deposit box changes her e-mail address.

So, she informs the new e-mail address to the deposit box agent. The agent, in turn, executes an update in the lessor register through a PATCH method:



Unlike the PUT method, the PATCH method allows the data update of particular fields of an entity. In our example, the deposit box lessor changed only the e-mail information, keeping the rest of the register with the same data. So, the PATCH method fits well to handle this specific update.

5. PUT vs. PATCH

In some scenarios, it is hard to decide on using PUT or PATCH. It occurs because, in these scenarios, both PUT and PATCH seem to achieve the same final result. For example, let's consider the PUT requesting shown in the third section. In this particular case, requesting a PATCH method instead of PUT will provide an equivalent result: all the fields of the requested entity are updated. However, PUT is idempotent by definition. So, PUT is a more fault-tolerant option than PATCH, making it a better choice in the considered case.

However, there exist scenarios where simply changing the requested method will produce different results. For example, requesting a PUT method instead of PATCH in the scenario of the fourth section will generate information loss. In this case, the PUT method will make everything except the e-mail data removed from the entity. Thus, only the PATCH method is adequate to meet the objectives of this specific request case.

The decision-making of employing a PUT or PATCH method should consider their particular characteristics. So, the following table presents some of these characteristics that can help in this process:

	PUT	PATCH
Request With Body Content	Yes	Yes
Successful Response With Body Content	No	Yes
Safe	No	No
Idempotent	Yes	No

6. Conclusion

In this article, we learned about the HTTP methods of PUT and PATCH. Initially, we had a brief review of the HTTP protocol. Then, we in-depth analyzed the PUT and PATCH methods through their theoretical descriptions and practical examples. At last, we compared both PUT and PATCH to outline scenarios where each one suits better.

We can conclude that the PUT and PATCH methods have many similarities. However, they have specific characteristics that must be considered while implementing an HTTP server and then requesting it.

CATEGORIES

ALGORITHMS

ARTIFICIAL INTELLIGENCE

CORE CONCEPTS

DATA STRUCTURES

GRAPH THEORY

LATEX

NETWORKING

SECURITY

SERIES

ABOUT

ABOUT BAELDUNG

THE FULL ARCHIVE

WRITE FOR BAELDUNG

EDITORS