(/cs/) (https://www.baeldung.com/cs/)

Difference Between REST and HTTP

Last modified: August 3, 2021

by Michael Pratt (https://www.baeldung.com/cs/author/michael-pratt)

Networking (https://www.baeldung.com/cs/category/networking)

Programming (https://www.baeldung.com/cs/category/core-concepts/programming)

Web (https://www.baeldung.com/cs/category/web)

1. Introduction

Often times the terms REST and HTTP are used interchangeably. In this article, we'll look at what each term really means and why they are two different things.

2. What Is REST?

REST stands for *Representational State Transfer*. It was first described in Roy Fielding's now-famous dissertation (https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm). In this paper, Fielding laid out his vision of an ideal **software architecture for the World Wide Web**.

REST is *not* a standard or a specification. Instead, Fielding described REST as an architectural style for distributed hypermedia systems. He believed there are several attributes of a RESTful architecture that would make it ideal for large interconnected systems such as the internet.

Next, we'll look at the attributes that define RESTful systems.

2.1. Resources and Representations

The core building blocks of RESTful systems are resources. A resource can be a web page, a video stream, or an image, for example. A resource can even be an abstract concept, such as the list of all users in a database or the weather forecast for a particular location. The only real constraint is that **every resource in a system is uniquely identifiable**.

Additionally, resources may be available in multiple representations. In a client-server model, **the server is responsible for managing the state of the resource, but a client can choose which representation they prefer to interact with**.

2.2. Uniform Interface

The uniform interface is one of the distinctive attributes of RESTful systems. **It requires clients to access all resources using the same set of standard operations**.

The benefits of a uniform interface are the ease of creating implementations that are decoupled from the services they provide. This allows services to evolve without impacting clients. The trade-off is that a uniform interface may impose unnecessary restrictions on some systems or require servers to perform less efficiently than they could with specialized operations.

2.3. Stateless

All operations in a RESTful system should be stateless. This means each call from the client to the server must not rely on any shared state. It also implies that every request to the server must include all required data for the server to fulfill the request.

The stateless requirement in REST gives way to several key properties:

- Visibility: Each request can be analyzed in isolation to monitor system health and responsiveness
- Reliability: System failures are easier to recover from
- Scalability: It's simple to add more server resources to handle more requests

However, it also comes with the trade-off that requests to the server are larger and contain repetitive data when a client has multiple interactions with the server.

3. What Is HTTP?

Unlike REST, **the HyperText Transfer Protocol (HTTP) is a standard with well-defined constraints**. HTTP is the communication protocol (https://tools.ietf.org/html/rfc2616) that powers most of our everyday interactions on the internet:

- Web browsers loading web pages
- Streaming a video
- Using a mobile device to turn off the lights in a home

So, is REST the same thing as HTTP? The short answer is no.

HTTP is a protocol that is maintained by the Internet Engineering Task Force. While it is not the same as REST, **it exhibits many features of a RESTful system**. This is not by accident, as Roy Fielding was one of the original authors of RFC for HTTP.

It’s important to remember that **the use of HTTP is not required for a RESTful system**. It just so happens that HTTP is a good starting because it exhibits many RESTful qualities.

Let’s take a closer look at some of the qualities that make HTTP a RESTful protocol.

3.1. URLs and Media Types

In the world of HTTP, resources are typically files on a remote server. These could be HTML, CSS, JavaScript, images, and all the other files that comprise modern web pages. **Each file is treated as a distinct resource that is addressable using a unique URL**.

HTTP is not just for files, though. **The term resource can also refer to arbitrary data on a remote server**: customers, products, configuration settings, and so much more. This is why HTTP has become popular for building modern APIs. It provides a consistent and predictable way to access and manipulate remote data.

Additionally, **HTTP allows clients to choose different representations for some resources**. In HTTP, this is handled using headers and a variety of well-known media types.

For example, a weather web site may provide both HTML and JSON representations for the same weather forecast. One is suitable for displaying in a web browser, while the other is suitable for processing by another system that archives historical weather data.

3.2. HTTP Methods

Another way in which HTTP adheres to the principles of REST is that **it provides the same set of methods for every resource**. While there are nearly a dozen available HTTP methods (https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods), most services deal primarily with the 4 that map to CRUD operations: *POST*, *GET*, *PUT*, and *DELETE*.

Knowing these operations ahead of time makes it easy to create clients that consume a web service. Compared to protocols such as SOAP (/java-soap-web-service), where operations can be customized and unlimited, HTTP keeps the known set of operations minimal and consistent.

Of course, individual web services may choose to deny certain methods for some resources. Or they require authentication for sensitive resources. Regardless, the set of *possible* HTTP methods are well known and cannot vary from one web site to another.

3.3. HTTP Is Not Always RESTful

However, for all the ways in which HTTP implements RESTful principles, there are multiple ways in which it can also violate them.

First, REST is not a communications protocol, while HTTP is.

Next, and perhaps most controversially, is that **most modern web servers use cookies and sessions to store state**. When these are used to refer to the state on the server-side, this violates the principle of statelessness.

Finally, **using URLs (as defined by the IETF) might allow a web server to violate the uniform interface**. For example, consider the following URL:

```
https://www.foo.com/api/v1/customers?id=17&action=clone
```

While this URL has to be requested using one of the pre-defined HTTP methods such as *GET*, it is using query parameters to provide additional operations. In this case, we're specifying an action named *clone* that isn't obviously available to all resources in the system. It's also not clear what the response would be without more detailed knowledge of the service.

4. Conclusion

While many people continue to use the terms REST and HTTP interchangeably, the truth is that they are different things. REST refers to a set of attributes of a particular architectural style, while HTTP is a well-defined protocol that happens to exhibit many features of a RESTful system.

CATEGORIES

ALGORITHMS (/CS/CATEGORY/ALGORITHMS)
ARTIFICIAL INTELLIGENCE (/CS/CATEGORY/AI)
CORE CONCEPTS (/CS/CATEGORY/CORE-CONCEPTS)
DATA STRUCTURES (/CS/CATEGORY/DATA-STRUCTURES)
GRAPH THEORY (/CS/CATEGORY/GRAPH-THEORY)
LATEX (/CS/CATEGORY/LATEX)
NETWORKING (/CS/CATEGORY/NETWORKING)
SECURITY (/CS/CATEGORY/SECURITY)

SERIES

DRAWING CHARTS IN LATEX (/CS/CATEGORY/SERIES)

ABOUT

ABOUT BAELDUNG (HTTPS://WWW.BAELDUNG.COM/ABOUT)
THE FULL ARCHIVE (/CS/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (HTTPS://WWW.BAELDUNG.COM/EDITORS)