WHAT IS IDEMPOTENCE?

Think about the “Pay now” button the last time you bought something online. Now, let’s say you’re ready to purchase your shopping cart items, so you press “Pay now”. What happens if you press it multiple times? Will your credit card be charged for each time you hit the button?

This is where idempotence comes in: if the function in the “Pay now” button is idempotent, the user can tap the “Pay Now” button many times—yet their card is charged only once and they only make one purchase on their cart’s contents. If, alternatively, the function of the “Pay Now” button weren’t idempotent and the user anxiously pressed the key several times, the user would end up with multiple charges on and receive several of the same items in their mailbox.

So, idempotence is an important design for many functions in technology. Let’s take a look.

Idempotence: a technical definition

Idempotence is **any function that can be executed several times without changing the final result beyond its first iteration**. Idempotence is a technical word, used in mathematics and computer science, that classifies a function’s behavior. There are certain identity functions, such as a = a, that can be called idempotent.

As a function, it can be expressed as:

```
f(f(x)) = f(x)
```

That means that an operation can be performed on x to return y. Then, if that same function were performed on y, the result would still equal y.

When to use idempotence

Idempotence is a safe practice for DevOps teams when developing applications. Idempotence ensures a safe, quality experience for both users and software teams. No one has to go in afterwards and clean up a mess.

For example, idempotence is the design standard for Ansible, a DevOps tool for system administrators to manage their servers. Ansible can automate the server setup process, consistently create the same servers, and automatically deploy them. There is a lot of orchestration that occurs in automating this process—and it is idempotent behaviors that ensure only one directory gets created on a server, and only an exact number of servers get created.

The idempotence function can be used in a variety of ways. Engineers want the destination to be the same, but the route that an orchestrator wishes to take can be different. If one function is called and the route it takes fails, another route can be tried. If one route lags, and a new one is fired, then, if both tasks make it to the endpoint, the result at the destination can still be the same.

Idempotent behavior is ever more crucial in cloud computing and edge devices. When multiple accounts can be logged in at once, accessed from different locations and different devices, updated from multiple user accounts, and data sent from one place to another, idempotent design helps manage the application's state more efficiently and without error.

Sample idempotent functions

Here are some idempotent functions in Python and HTTP:

Idempotent Python functions

Some idempotent functions in Python include:

```
abs(abs(x)) = abs(x)

my_set.discard(x)

Y = x * 0
```

If absolute value is taken on -12, the result is 12. If the absolute value were again taken of the result, 12, the function returns 12.

Idempotent HTTP functions

In the land of HTTP methods, the GET, DELETE, and PUT functions are idempotent.

The GET method retrieves information from an HTTP endpoint. Whatever information GET asks for will be the same information again and again. This behavior is illustrated by hitting refresh on an internet browser. The browser pings an IP address and gets information. If GET weren't idempotent, perhaps it would stack more and more pages on top of each other, adding each refreshed page in

the browser's cache. DELETE has similar behavior.

PUT is also an idempotent function—but POST is not. In HTTP, PUT will define a value, and continue to define the value as whatever is stated.

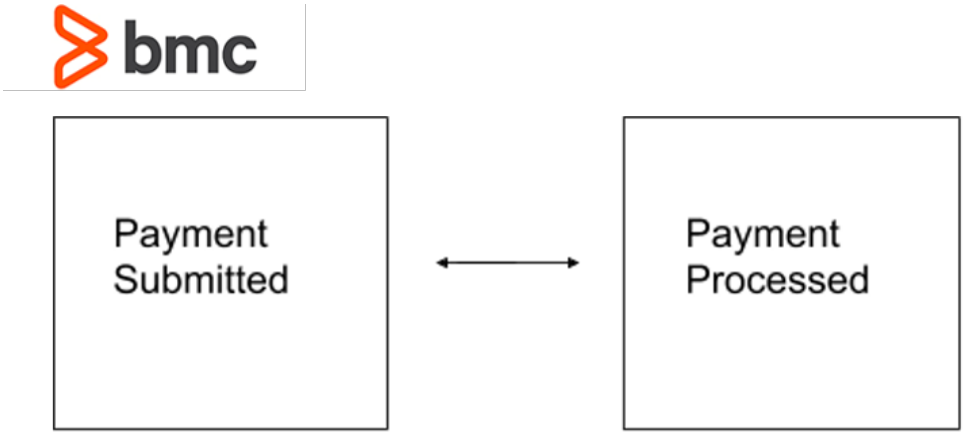
POST works differently. POST can send the same value repeatedly, but it adds an element to a collection. POST could repeatedly send the same Twitter status 10 times per hour, and all the followers will receive 10 status updates all saying the same thing. The PUT function just puts the status out there once.

Example of idempotence in front end design

Idempotence is achieved in practice when setting how data moves from one place to another in the IT infrastructure. Containerization of front-end applications helps split the function of applications from one-task-handles-all, to several different operations. This creates a more stable design

Non-idempotent example

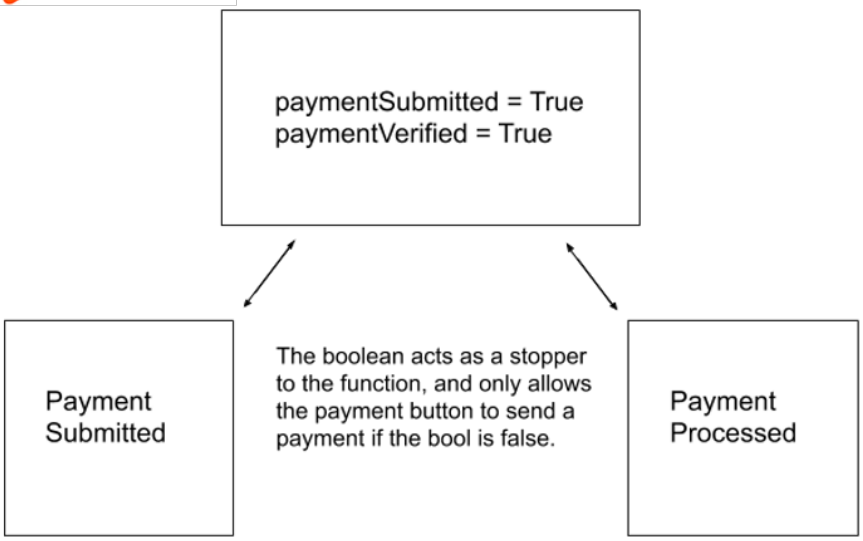
Let's take our "Pay now" example from above. In a non-idempotent design, a submit payment method can attempt to contact a bank account directly to make a payment. So, each time you click the "Pay now" button, that single press activates a function to take your information and submit it to the credit card service. The credit card service can return success or failure, and the user receives a either a success message or a payment failed message.



Non-idempotent design

Idempotent example

If this were to be turned into an idempotent design, the function would pass through an object before being submitted to the credit card service. Instead of contacting the credit card service directly, the submit button might store the user information in a data model, in which one of the values says paymentSubmitted = True and one value says paymentVerified = False.



Idempotent design

By creating a variable paymentSubmitted = True, and checking this value before submitting the card information, when the user presses the “Pay Now” button multiple times, it will only go through once when the Submitted variable equals false and all other presses will be null, as the function waits for its initial submission to return.

Additional resources