#ASLI ENGINEERING

Designing Idempotent Payments API

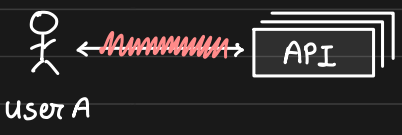
SWIPE

BY

ARPIT BHAYANI

Designing Idempotent APIs

Say user A makes an API call to the backend and there was some issue with network and hence the call failed!

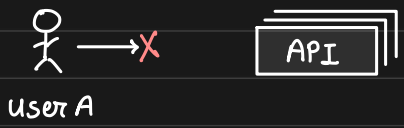


What do we do ?

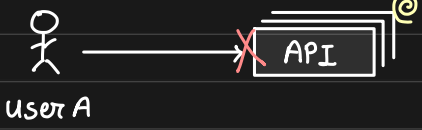
1. We ignore and move forward
2. We pass the observed error to the user
3. We retry on our own
   * better user experience hence preferred

Depending on what we are building, we pick one of these ways to handle

Failure Timeline



Client knows that the request didn't even reach the server, hence, safe to retry



Failure while server is executing. Client cannot be sure of retrying.

Server completed execution but before the response reach client, n/w failed

Client cannot sure of retrying.

What could go wrong if we always retry?

Say, the API was about transferring money from A to B

`/payment/<B>` `{ amount: 10000 }`

If the API call was processed successfully, but we still retried, then the amount will be re-deducted.

for n retries, the amount deducted = $10000 x n

Payments and other critical APIs needs to ensure idempotency so that we safely retry.

Idempotent API

Core idea of building an idempotent API is that the server should somehow know that it has seen this request earlier.

if seeing it the 1st time → proceed
otherwise → ignore / throw error

What we are trying to achieve is **Exactly Once Semantics**

Disambiguating request on the basis of simple URL is not a good idea

- ↳ How many URLs would we keep track of? ↳ High mem consumption
- ↳ What if request is genuine?

The way to achieve this is **Idempotence Keys**

1. Client first talks to the server to generate a random ID
   ↳ the ID may be operation specific eg: 'Money Transfer'
2. Client passes this ID along with regular payload to do actual operation
3. Server checks the ID and validates if it has already handled it
   if already handled it → ignore / throw error
   if not → handle it
4. if client sees the error, then it retries the request with same ID
5. Server extracts the ID, validates, and then
   decides to handle it, resume, or ignore → use case specific

How do we pass the idempotency key?

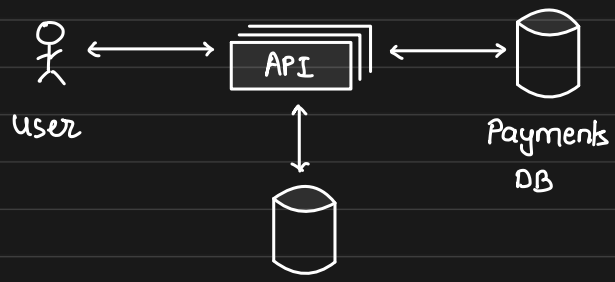
1. Request Header
2. Query Parameter

} most common

Stripe requires clients to pass Idempotency keys in Request Header "Idempotency-Key".

Architecture

1. Server maintains all idempotency keys in a database
2. When operation is successful,
   the server deletes the key
3. For every request, server checks the DB for the key



key1 : { operation, user, ttl }
key2 : { operation, user, ttl }
key3 : { operation, user, ttl }