**Task 4 : Rate Limiting Approaches**

Implementation of basic rate limiter in Python : [Github](https://github.com/vishvas04/Zeta_Assignment/blob/main/Backend%20API%20Design%20-%20Rate%20Limiting%20for%20Banking%20Transactions/Rate_Limiting_Zeta.ipynb)

This method as far as I know is used to avoid DDOS attacks and also it restricts users from making continuous requests.

It becomes particularly valuable when scaling full-stack applications, where too many users make requests to the servers which makes genuine users locked out due to resource constraints.

We use Rate Limiting Approaches to limit this problem!

**1) Fixed Window Counter**

This algorithm divides time into fixed intervals (e.g., 1-second windows) and tracks the number of requests a user makes within each interval. For example, if the limit is set to 5 requests per second, the system allows up to 5 requests in each 1-second window and resets the count at the start of the next interval. With this approach we can limit the users in making utmost 5 requests for 5 seconds. This approach is straightforward to implement and memory-efficient requiring only a counter per user.

A key drawback is when the traffic increases the window size goes out of boundary. For instance, a user could send 5 requests at the end of one window and another 5 immediately at the start of the next, within a short span. Additionally, fixed windows cannot identify indefinite traffic patterns.



**2) Token Bucket**

The token bucket algorithm assigns each user a virtual “bucket” that holds a predefined number of tokens. Tokens are given again if it gets extinct at a steady rate(e.g., 5 tokens per second), and each request consumes one token.

If the bucket is empty, requests are either delayed or rejected until new tokens are added. This approach ensures smoother traffic control by distributing requests more evenly and allowing users to “save” unused tokens for future use. However, it introduces complexity, as the system must continuously track token levels and handle token refills.

**Trade-Offs Between Approaches**

**1) Fixed Window Counter**

* Simple and easy to set up.
* Uses little memory since it only tracks request counts.
* Can cause short bursts of extra traffic when limits reset (e.g., making 5 requests at the end of one window and 5 more at the start of the next).
* Not flexible for changing traffic patterns.

**2) Token Bucket**

* Helps control traffic smoothly by refilling tokens gradually.
* Allows users to save unused tokens for later use.
* Harder to implement because it requires constant tracking of token levels.
* Needs extra processing power to manage token updates.