Microservice Rate Limiting System Design

Functional Requirements

Limit each client (by IP or token) to a fixed number of requests per time window

Block excessive requests with an HTTP 429 response

Apply rate limiting globally or per endpoint as needed

Log and monitor rejected requests

Non-Functional Requirements

Minimal latency overhead (should not slow down response time)

Lightweight for single-service deployment (no extra infra required)

Optional support for distributed limit coordination (e.g., Redis)

Easy to configure and maintain

Focus is on **Consistency and Availability**. It's important that request counts are accurate and consistently enforced across all replicas — otherwise abuse may slip through. Partition Tolerance is less critical, since this service runs best within a single region or tightly connected cluster.

Chosen Algorithm: Token Bucket

Allows bursts while enforcing average request rates

Replenishes tokens at a fixed rate over time

Denies request when bucket is empty (returns 429)

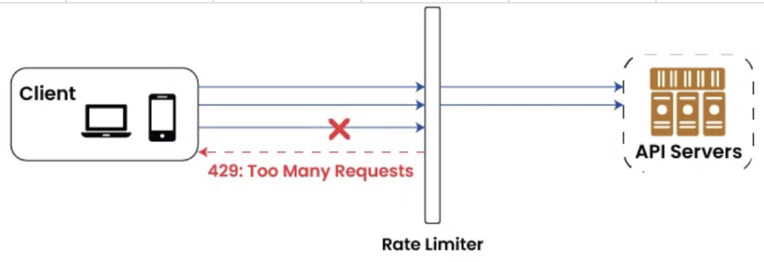
Architecture Overview

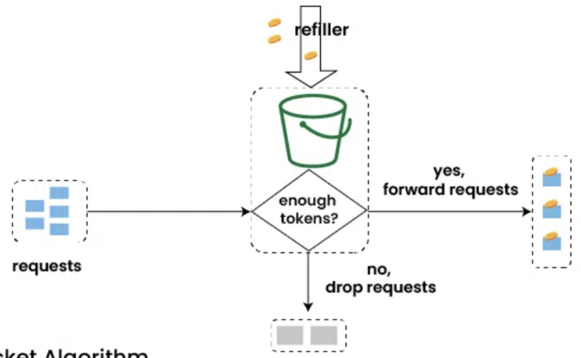
Clients	Any API consumer (browser, mobile app, back
Entry Layer	FastAPI App with endpoint /api/v1/data All requests pass through rate limiter
Rate Limiter Middleware	Integrated via slowapi Keeps token count per client/IP Refill rate: 1 token/second Max tokens: 100 (100 requests per 100 sec)
Application Logic	If allowed: returns JSON success response If limited: returns HTTP 429 with error messag
Optional	Redis (if scaling across multiple instances) Prometheus/Grafana for monitoring

Flow Diagram (Text Format)

```
Client → FastAPI App → RateLimiter Middleware →
   ├──     Enough tokens → Handle request → 200 OK
   └──     No tokens left → Return 429 Too Many Requests
```

Link to GitHub:
https://github.com/jaxylykovm/rate-limit-microservice





Token Bucket Algorithm