Rate Limiter:
Why:
to prevent DoS(Denial of Service) attack
to reduce cost(esp for paid 3rd party apis)
to reduce server load

Basics:
Client Side or Server Side ? >> Server Side
Implemented on Server Side or in a Gateway ? >> Your design decision
Throttling based on what ? user id, ip address, …
Scaling System ? for start up, for big company with large user base ?
System to work in a Distributed environment ? >> Yes
Will inform throttled user ?(Exception Handling) >> Yes

Algorithms: fixed window, sliding window, token bucket

HLD:
Since Rate Limiter is adding extra layer on top of the existing system, must be really FAST
Simple overview of the System: we've USER and COUNTER to track per User
So, where to store this COUNTER ? DB is slow for our need, so In Memory Cache is good since it is fast and it supports time based expiry strategy.
Ex. Redis Commands: INCR and EXPIRE

How Rules are created ? Where to store them ? (Cache + Original DB) is good
Rules are generally saved in configuration files and saved on Disk.
How to handle rate limited users ? 429 code response, and also good to send every OK response header with information like how many hit already done, or how many hit left etc. to make user experience better.
Two options for rejected requests: Just Drop it, or Push it to Queue to process it later which is important like in ordering/purchasing something but failed due to traffic volume.

Distributed Environment:
Single Server is Easy
Supporting multiple servers and concurrent Threads needs more work on race condition & Synchronization Issues.

Race Condition:
When multiple different threads access and modify shared resources without cookies or sticky sessions for the LB to direct User to the same server.
But cookies and sticky sessions which break scalability, flexibility and resilience as they tie users to a specific server which undermine the very things that make distributed systems powerful.
By flexibility, we mean able to change your infrastructure (like scaling down, scaling horizontally), deploy updates, or reroute traffic without worrying about USER STATE. Sticky

sessions get in the way for all these. The better solution is CENTRALIZED data stores like Redis to keep whatever data you keep in sticky sessions.

Race Condition Solution:

1. Lock (but slows down the system)
2. Lua Script and Sorted Sets data structure in Redis
   Since Redis is < 128GB, we may need Sharded Redis

Lua Script behaves like an implicit lock around your logic, but without any of the overhead of an explicit distributed lock which is slower.

Summarizing distributed system:
Multiple Rate Limiter Servers (Leader and followers)
Each Rate Limiter connected with one of the Redis replicas via Load Balancer
Each Rate Limiter also connected with the API Servers via Load Balancer

Monitoring and Observability:
Each Rate Limiter pushing data to the Monitoring Tool