Software Requirements Specification (SRS) Template

Project: API Rate Limiter System
Version: 1.0
Authors: QuadCore
Date: 20-08-2025
Status:

Revision History

Version	Date	Author	Summary	Approval
1.0	20-08-2025	QuadCore	API Rate Limiter	

Approvals

Role	Name	Signature / Email	Date
Course Coordinator	Mr. Arvind Srinath K		

Table of Contents

1. Introduction

1.1 Purpose

This SRS describes the requirements for an API Rate Limiter middleware/service that enforces rate limits on API endpoints, protecting backend services from abuse, DoS attacks, and ensuring fair API usage across clients.

1.2 Scope

The Rate Limiter will operate as middleware between API clients and backend services. It will restrict requests per user/IP within defined time windows, return appropriate error responses, and support configurable strategies (fixed window, sliding window, token bucket). Excludes core API logic.

1.3 Audience

- DevOps Engineers
- Frontend and Backend Developers
- API Consumers
- Security Analysts
- QA/Test Engineers

1.4 Definitions

- API: Application Programming Interface
- Rate Limiting: Restricting request frequency
- Token Bucket / Leaky Bucket: Algorithms used for limiting
- REST: Representational State Transfer
- HTTP: Hyper Text Transfer Protocol
- DoS Attack: Denial of Service

2. Overall Description

2.1 Product Perspective

The Rate Limiter plugs into existing API stacks as middleware or runs as a proxy service. It intercepts incoming API requests, checks current usage against pre-configured limits, and allows/blocks requests accordingly. The system exposes metrics and admin endpoints for health and statistics. Compatibility with popular API frameworks (REST) is targeted.

2.2 Major Product Functions

- Track API usage per user/IP
- Apply limits based on configuration
- Respond with HTTP 429 when limits exceeded
- Provide logs and monitoring

2.3 User Roles

- API Client: Any app/user making requests
- Admin: Configures rate limits, monitors logs

2.4 Operating Environment

Cloud/On-prem servers, works with REST APIs over HTTP/HTTPS.

2.5 Constraints

- Must work with stateless distributed servers.
- Limits must be enforced with <100ms overhead.

3. External Interfaces

3.1 User Interfaces

- feedback through API error responses (JSON)
- Admin/configuration through RESTful endpoints (secured)
- Logs for monitoring, alerting

3.2 Hardware Interfaces

N/A (software-only system).

3.3 Software Interfaces

Works with existing APIs via middleware integration. Logging/Monitoring API for admin dashboards.

3.4 Communication

REST/HTTPS. Error code: 429 Too Many Requests.

4. System Features

4.1 Core Features Table

Req ID	**Requirement**	**Description**	**Priority**	**Acceptance Criteria**
RL-F-001	Enforce API rate limits	System must enforce configured limits per user/IP/client.	High	Requests beyond the limit return HTTP 429 with Retry-After.
RL-F-002	Configurable limit windows	Admins can define limits per second/minute/hour/day.	High	Limits correctly applied per configuration.
RL-F-003	Logging of events	Log each request exceeding limit with timestamp, client ID, and endpoint.	High	Logs visible in admin dashboard.
RL-F-004	Admin REST API	Provide secured admin endpoints for viewing/modifying limits and fetching stats.	Medium	Admin can update limits and view live stats.

Req ID	**Requirement**	**Description**	**Priority**	**Acceptance Criteria**
RL-F-005	Distributed counter backend	Support scalable storage like Redis or Memcached for rate tracking.	Medium	Passes high concurrency tests.
RL-F-006	Retry-after header	Include Retry-After header and error message in HTTP 429 responses.	High	Header present and value correct.
RL-F-007	User authentication integration	Rate limits should apply after user authentication to support user-level throttling.	Medium	Verified through test API calls.
RL-F-008	IP-based throttling	Apply rate limits per IP address when no user identity is provided.	High	Requests from same IP are counted correctly.
RL-F-009	Configuration import/export	Admin can export/import configuration as JSON/YAML.	Low	Configuration persistence tested.
RL-F-010	Monitoring and metrics	Expose metrics (requests blocked, allowed, latency) for Prometheus/Grafana dashboards.	Medium	Metrics endpoint available and accurate.
RL-F-011	Health check endpoint	Provide /health endpoint to verify service uptime.	Medium	Returns 200 OK with status message.
RL-F-012	Alerting mechanism	Trigger alert/log when error rate exceeds threshold.	Medium	Alerts visible via logs or monitoring tool.

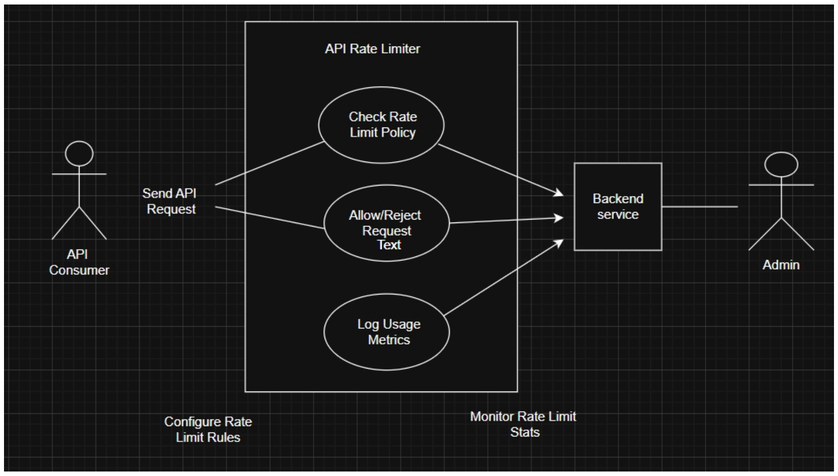
5. Non-Functional Requirements

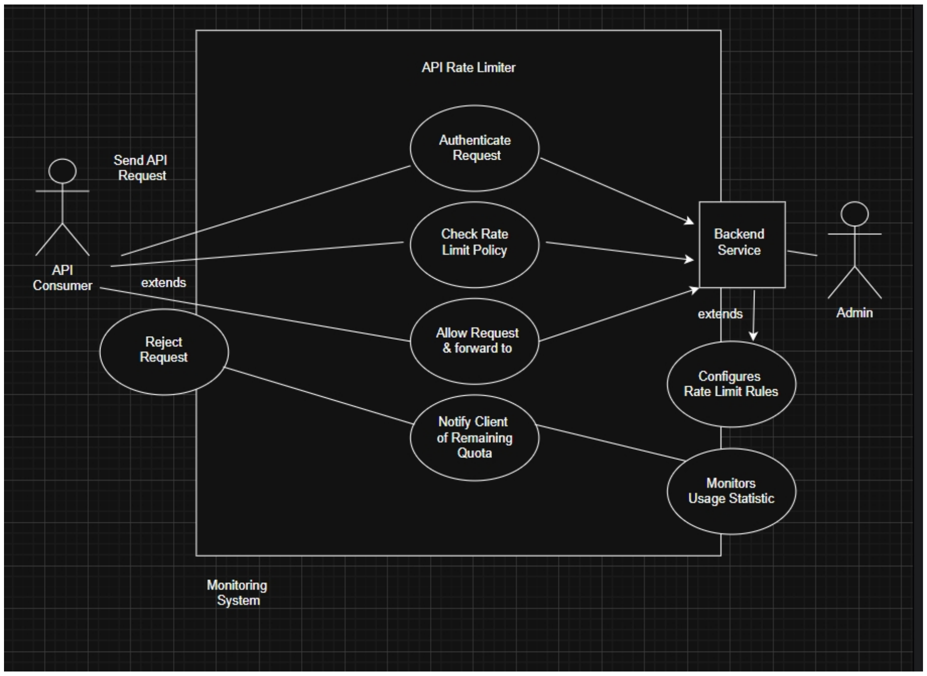
Req ID	Requirement	Category	Priority	Acceptance Criteria
RL-NF-001	Latency overhead ≤1 ms at p99	Performance	High	Profiling confirms response latency within limit.
RL-NF-002	System uptime ≥99.99%	Reliability	High	Monitored uptime reports show compliance.
RL-NF-003	Secure admin access (RBAC, authentication)	Security	High	Unauthorized requests rejected; access logged.
RL-NF-004	Data protection and audit	Security/Compliance	High	Config/logs encrypted, audit logs maintained.
RL-NF-005	GDPR compliance and log retention	Privacy	Medium	Logs retained ≤1 year; auto-purge tested.
RL-NF-006	Scalability	Performance	High	Supports horizontal scaling across nodes.

6. Quality Attributes & Acceptance Tests

- Performance: Meet stated latency and throughput targets.
- Reliability: Auto-recover/alert on counter backend failures.
- Security: Prevent privilege escalation, config tampering, or information leaks.
- Scalability: Scale horizontally to support API clusters.

7. UML Use-Case Diagram





8. Requirements Traceability Matrix (RTM)

Req ID	Requirement short	Section ref / Design Spec	Module	Test case(s)	Status	Comments
RL-F-001	Enforce rate limit	4.1 / DS-Limit-01	RateLimiter	TC-RL-01	N	

Req ID	Requirement short	Section ref / Design Spec	Module	Test case(s)	Status	Comments
RL-F-004	Admin REST API	4.1 / DS-Admin-01	AdminAPI	TC-Admin-01	N	
RL-NF-005	p99 latency target	5 / DS-Perf-01	All	TC-Perf-01	N	
RL-NF-007	Secure configs/logs	5 / DS-Sec-01	ConfigManager	TC-Sec-01	N	