

Software Requirements Specification (SRS)

Project: API Rate Limiter System

Version: 1.0

Authors: < Instructor / Students : Raunak Bagaria, R Jeevith, Rishav Ghosh, Roshit Sharma >

Date: 22-08-2025

Status: Approved

Revision history

| Version | Date | Author | Change summary | Approval |
|---------|------------|----------------|---|----------------|
| 1.0.1 | 21-08-2025 | R Jeevith | Overall Description , External Interface Requirements | Raunak Bagaria |
| 1.0.2 | 21-08-2025 | Raunak Bagaria | Functional requirements | R Jeevith |
| 1.0.3 | 21-08-2025 | Rishav Ghosh | Introduction, Requirements Traceability Matrix | Roshit Sharma |
| 1.0.4 | 22-08-2025 | Raunak Bagaria | System features, Non functional and Security requirements | R Jeevith |
| 1.0.5 | 22-08-2025 | Roshit Sharma | Quality and acceptance, UML diagram | Rishav Ghosh |
| 1.0.6 | 22-08-2025 | Rishav Ghosh | Requirements Traceability Matrix | Raunak Bagaria |

Approvals

| Role | Name | Signature / Email | Date |
|--------------------|------|-------------------|------|
| Course Coordinator | | | |

Table of Contents

1. Introduction
2. Overall description
3. External interfaces
4. System features (detailed)
5. Non-functional requirements (detailed)
6. Quality attributes & Acceptance tests
7. UML Use-Case Diagram
8. Requirements Traceability Matrix (RTM)

1. Introduction

1.1 Purpose

The primary purpose of the system is to serve as a middleware or standalone service that enforces rate limits on API endpoints to prevent abuse, ensure fair usage, and safeguard backend services from excessive traffic and denial-of-service attempts. The system ensures backend services are protected from abuse such as excessive request traffic or denial-of-service attacks, while providing fair usage for all clients.

1.2 Scope

The API Rate Limiter operates as an intermediary layer between clients and backend services. It monitors incoming API requests and applies configurable rate-limiting strategies—such as fixed window counters, sliding windows, token buckets, and leaky buckets—to regulate access. When request thresholds are exceeded, the system responds with standardized feedback to inform clients of quota usage and reset intervals.

1.3 Audience

- Developers – People who will build and improve the system, making sure it works as expected.
- Users of APIs – Applications or services that will go through the rate limiter and need fair access without overloading the backend.
- Administrators – Those who will set the rate-limiting rules, check logs, and keep the system running smoothly.
- Stakeholders – Anyone who owns or depends on the APIs, and wants to make sure they stay reliable and secure.
- Evaluators – People who will review the project to see if it meets the goals like scalability, security, and reliability.

1.4 Definitions

API – Application Programming Interface

Rate Limiting – The controlled restriction of API request rates within defined time intervals

Token Bucket / Leaky Bucket – Algorithms used to implement rate-limiting strategies

Client – An application, user, or service consuming backend APIs

Administrator – A system role responsible for managing rate-limiting policies, monitoring, and exception handling

2. Overall description

2.1 Product perspective

The API Rate Limiter is a middleware service that sits between client applications and backend APIs. Its purpose is to regulate the number of requests clients can make in a given period of time. By enforcing limits, it protects backend services from overuse, ensures fair access for all clients, and prevents abuse such as denial-of-service attacks. The system can run as a standalone service or be integrated into existing API gateways.

2.2 Major product functions (detailed)

- Enforce rate limits using different strategies (fixed window, sliding window, token bucket, leaky bucket).
- Support multiple scopes for limiting (per user, per IP address, per API key, per endpoint).
- Provide clear responses to clients when limits are exceeded (HTTP 429 with standard headers).
- Allow administrators to configure and adjust limits dynamically.
- Log and report usage statistics for auditing and monitoring.
- Maintain consistency of limits in distributed deployments.
- Handle failures gracefully with configurable fallback behavior.

2.3 User roles and characteristics (expanded)

- **API Consumer:** Application or developer calling APIs. Expects predictable performance and clear feedback when hitting limits.
- **API Provider:** The backend service being protected. Relies on the rate limiter to maintain system stability and availability.
- **Administrator / Operator:** Responsible for configuring policies, monitoring performance, and managing exceptions. Requires usable tools for policy management and logs.

2.4 Operating environment

- Runs in containerized environments (Docker/Kubernetes) or directly on servers.
- Works with REST or gRPC APIs over HTTPS.
- Uses in-memory stores like Redis or Memcached for request counters.
- Integrates with monitoring and logging systems (Prometheus, ELK).
- Supports scaling horizontally with multiple nodes.

2.5 Constraints

- Must keep latency overhead low (target under 10ms per request).
- Must operate securely using TLS for all external communication.
- Dependent on the availability and performance of the backing datastore.
- Configurable trade-off between strict consistency and high availability in distributed setups.
- Must handle network failures and datastore outages with predictable behavior.

3. External interface requirements

3.1 User interfaces

- **API Responses:** When a client exceeds limits, the system responds with HTTP 429 and includes clear error messages.
- **Response Headers:** Provide standard information such as `X-RateLimit-Limit`, `X-RateLimit-Remaining`, `X-RateLimit-Reset`, and `Retry-After`.
- **Administration:** Configurable through admin API or configuration files for defining and updating rate-limiting rules.
- **Optional Dashboard:** A simple web-based UI may be provided for visualization of usage and limits.

3.2 Hardware interfaces

Not applicable directly, as this is a software-based system. Requires hosting on servers or containers with sufficient CPU, memory, and network capacity.

3.3 Software interfaces

- **Protected APIs:** The rate limiter regulates requests to backend APIs (REST, gRPC)..
- **Admin API:** Exposes endpoints to manage rate-limiting policies, query usage, and adjust configurations.
- **Monitoring/Logging:** Provides metrics (requests allowed, blocked, latency) and logs for integration with monitoring systems.
- **Persistence Layer:** Connects to Redis, Memcached, or a similar datastore for request tracking.

3.4 Communications

- **Protocols:** Operates over HTTP/HTTPS for client and admin communication..
- **Response Feedback:** Returns JSON error messages and informative headers for throttled requests.
- **Distributed Coordination:** Uses storage or messaging systems (e.g., Redis Pub/Sub) for synchronizing limits across nodes.
- **Security:** All communications must be encrypted (TLS 1.2+), and administrative access must be authenticated and authorized.

4. System features (detailed)

Each requirement below includes acceptance criteria and a reference test case. IDs follow F-###.

4.1 Client Identification

Description : Identify and classify clients making API requests using various methods.

| Req ID | Requirement | Type | Priority | Source/ Stakeholder | Acceptance criteria / Test case ref | Comments / Dependencies |
|--------|--|------------|----------|------------------------|--|-----------------------------------|
| F-001 | The system shall identify clients using API keys. | Functional | High | API Admin | AC-F-001: Client correctly identified and classified. Test: TC-ID-01 | Requires token validation library |
| F-002 | The system shall support IP-based identification with CIDR range matching. | Functional | High | API Admin | AC-F-002: IP ranges correctly matched and classified. Test: TC-ID-02 | Requires network utilities |
| F-003 | The system shall support IP-based allowlists and blocklists. | Functional | High | Security | AC-F-003: Listed IPs processed according to rules. Test: TC-ID-03 | |

4.2 Rate Limit Enforcement

Description : Apply configured rate limits and take appropriate action when limits are exceeded.

| Req ID | Requirement | Type | Priority | Source/ Stakeholder | Acceptance criteria / Test case ref | Comments / Dependencies |
|--------|--|------------|----------|------------------------|--|--|
| F-004 | The system shall enforce rate limits per client across multiple time windows (second, minute, hour, day) | Functional | High | User | AC-F-004: Requests blocked with HTTP 429 Too Many Requests, when limits exceeded within any configured window. Test: TC-RL-01 | Core Policy |
| F-005 | The system shall support different rate limiting algorithms (token bucket, sliding window, fixed window) | Functional | Medium | User | AC-F-005: All algorithms correctly implemented and selectable. Test: TC-RL-02 | Time window management |
| F-006 | The system shall allow different limits per endpoint, IP, or API key. | Functional | High | User, API Admin | AC-F-006: Admin can perform CRUD on policies. Test: TC-RL-03 | Customization, tiered system for users |
| F-007 | The system shall forward approved requests to the target API servers while preserving original request headers and body content. | Functional | High | User | AC-F-007: Approved requests forwarded with 100% header preservation and response returned within SLA timeframes. Test: TC-RL-04 | Core Policy |

4.3 Analytics and Monitoring

| Req ID | Requirement | Type | Priority | Source/ Stakeholder | Acceptance criteria / Test case ref | Comments / Dependencies |
|--------|---|------------|----------|------------------------|--|-------------------------------|
| F-008 | The system shall trigger alerts when clients exceed 80% of their rate limits. | Functional | High | Operations | AC-F-008: Alerts sent within 1 minute of threshold breach. Test: TC-AM-01 | Notification system for users |
| F-009 | The system shall support custom response headers for rate limit information. | Functional | Low | User | AC-F-009: Headers include remaining requests, reset time, and limit values. Test: TC-AM-02 | |
| F-010 | The system shall display current request counts for each client on admin dashboard. | Functional | Low | API Admin | AC-F-010: Dashboard shows real-time counters updated every 30 seconds. Test: TC-AM-02 | Admin Interface |

4.4 Configuration Management

| Req ID | Requirement | Type | Priority | Source/ Stakeholder | Acceptance criteria / Test case ref | Comments / Dependencies |
|--------|---|------------|----------|------------------------|---|----------------------------|
| F-011 | The system shall support dynamic rate limit rule updates without service restart. | Functional | High | Operations | AC-F-010: Rules updated within 30 seconds across all instances. Test: TC-CM-01 | Configuration propagation |
| F-012 | The system shall validate configuration syntax and semantics before applying changes. | Functional | High | Operations | AC-F-011: Invalid configurations rejected with clear error messages. Test: TC-CM-02 | Configuration validation |

| | | | | | | |
|-------|--|------------|--------|-------------|---|----------------|
| F-013 | The system shall support rule precedence with client-specific overrides. | Functional | Medium | Business | AC-F-012: Client-specific rules take precedence over global rules. Test: TC-CM-03 | Rule hierarchy |
| F-014 | The system shall allow API admin to temporarily block a source manually. | Functional | High | Admin | AC-F-014: All requests from the blocked source shall be denied (returning HTTP 403 Forbidden) until the configured time expires. Test: TC-CM-04 | Admin Control |
| F-015 | The system shall allow API admin to define and display a custom error message. | Functional | Low | Admin, User | AC-F-015: When a request is blocked, the configured custom message appears in the response body along with HTTP status. Test: TC-CM-05 | Customization |

5. Non-functional requirements (detailed)

NFRs below are measurable and tied to test plans. IDs follow NF-###.

| Req ID | Requirement | Category | Priority | Acceptance criteria/Measurement |
|--------|---|-------------|----------|--|
| NF-001 | Rate limiter overhead should not add more than 10ms | Performance | High | 95th percentile latency <= 10ms in production-like |

| | | | | |
|--------|--|----------------|------|---|
| | latency to API requests. | | | e test. Test: TC-Perf-01 |
| NF-002 | System shall provide 99.9% availability. | Reliability | High | Uptime monitoring shows atleast 99.9% monthly availability. Test: TC-Rel-01 |
| NF-003 | The system shall allow for storage and retrieval of request count data, per client and endpoint. | Data Retention | High | Metrics updated within 5 seconds of request processing. TC-Ops-01 |
| NF-004 | System shall maintain rate limit accuracy within 5% deviation. | Accuracy | High | Rate limit enforcement measured within 5% of configured limits. Test: TC-Acc-01 |
| NF-005 | The system must be able to handle atleast 1000 requests per minute. | Performance | High | Less than 1% failed requests during peak load. Test: TC-Perf-02 |

5.1. Security Objectives

1. **Prevent API Abuse:** Protect backend services from malicious or excessive requests that could degrade performance or cause service disruption.
2. **Maintain Data Integrity:** Ensure all rate limiting configurations, client data, and system logs remain tamper-proof.
3. **Protect Against Unauthorized access:** To prevent exploitation of sensitive system metrics, client usage patterns, and configuration details.

5.1. Security Requirements

1. SEC-001: All administrative interfaces require multi-factor authentication and role-based authorization.
2. SEC-002: Rate limit bypass attempts must be logged with full request context and trigger automated security alerts.
3. SEC-003: All sensitive configuration data including API keys and tokens must be encrypted at rest using AES-256 encryption.
4. SEC-004: System shall implement request signature validation to prevent configuration tampering and ensure data integrity.
5. SEC-005: Client IP addresses and personal identifiers must be anonymized in analytics data beyond 30 days operational retention period

6. Quality attributes & Acceptance test.

Exit Criteria for Acceptance

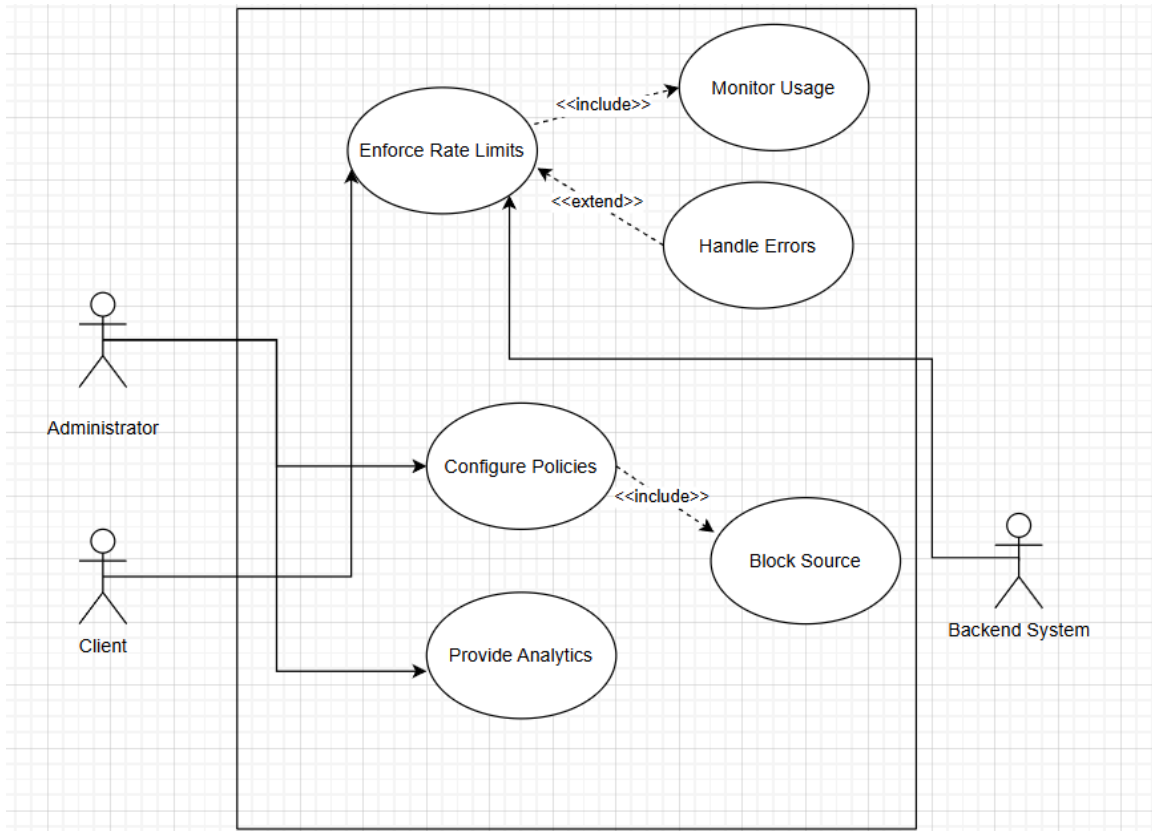
- Performance Benchmarks (NF-005): Added to ensure the system handles 1000 requests/min, a high-priority NFR.
- Security Standards: Implied from NF-001 context (low latency with secure protocols), ensuring compliance with security needs (e.g., F-003 IP blocklists).
- Thresholds (100%/90%): Makes "all test cases passed" measurable, reflecting high-priority requirements (e.g., F-001, F-004) vs. medium-priority (e.g., F-005, F-009).

Acceptance Test Suites

- Rate Limiting Enforcement: Replaces "Withdrawal/Deposit/Transfer" (ATM-specific) with a relevant suite for F-004 to F-007.
- Monitoring & Analytics: Covers F-008 to F-010, addressing operations and admin needs.
- Configuration Management: Addresses F-011 to F-015, ensuring dynamic updates and blocking capabilities.
- Error Handling: Added for F-007 (HTTP 429) and F-014 (HTTP 403), critical for client feedback.
- Reliability: Added for NF-002 (99.9% uptime), a key NFR.

7. System models and diagrams

UML Use-Case diagram



8. Requirements Traceability Matrix (RTM)

| Req ID | Requirement (Short) | Section Ref | Module | Test Case(s) | Status (N/P/A) | Comments |
|--------|---------------------------------|-------------|-----------|--------------|----------------|---|
| F-001 | Identify clients using API keys | 4.1 | Auth & ID | TC-ID-01 | N | I Need to find a good token parsing library for this. |
| F-002 | Support IP-based identification | 4.1 | Auth & ID | TC-ID-02 | N | Requires some network utils. |
| F-003 | Support IP | 4.1 | Auth & | TC-ID-0 | N | For security |

| | | | | | | |
|--------|---------------------------------------|-----|--------------------|------------|---|--|
| | allow/block lists | | ID | 3 | | requirements. |
| F-004 | Enforce rate limits (multi-window) | 4.2 | Core Limiter Logic | TC-RL-01 | N | This is the main logic. |
| F-005 | Support different limiting algorithms | 4.2 | Core Limiter Logic | TC-RL-02 | N | Medium priority, we can start with one and add more later. |
| F-006 | Allow limits per endpoint/IP/key | 4.2 | Core Limiter Logic | TC-RL-03 | N | For custom plans for different users. |
| F-007 | Forward approved requests | 4.2 | Proxy | TC-RL-04 | N | Must preserve all the headers. |
| F-008 | Trigger alerts on 80% limit breach | 4.3 | Monitoring | TC-AM-01 | N | Will need a notification system for this. |
| F-009 | Support custom response headers | 4.3 | Monitoring | TC-AM-02 | N | Low priority but good for users. |
| F-010 | Show request counts on dashboard | 4.3 | Monitoring | TC-AM-02 | N | Will be part of the admin UI. |
| F-011 | Dynamic rule updates (no restart) | 4.4 | Admin Config | TC-CM-01 | N | This is a high priority ops feature. |
| F-012 | Validate configuration syntax | 4.4 | Admin Config | TC-CM-02 | N | Important to avoid breaking the system. |
| F-013 | Support rule precedence | 4.4 | Admin Config | TC-CM-03 | N | e.g. client rules > global rules. |
| F-014 | Allow manual source blocking | 4.4 | Admin Config | TC-CM-04 | N | Emergency admin control. |
| F-015 | Allow custom error messages | 4.4 | Response Module | TC-CM-05 | N | .Customization |
| NF-001 | Latency overhead < 10ms | 5 | Performance | TC-Perf-01 | N | Has to be fast. |
| NF-002 | Provide 99.9% availability | 5 | Reliability | TC-Rel-01 | N | High priority NFR. |
| NF-0 | Store and | 5 | Data | TC-Ops- | N | Need for the |

| | | | | | | |
|--------|--|---|-------------|------------|---|---------------------------------------|
| 03 | retrieve request data | | | 01 | | monitoring part. |
| NF-004 | Maintain rate limit accuracy ($\pm 5\%$) | 5 | Accuracy | TC-Acc-01 | N | Limits cannot be off by more than 5%. |
| NF-005 | Handle >1000 requests/min | 5 | Performance | TC-Perf-02 | N | Load testing target. |