Business Requirements Document (BRD): Rate Limiter Dashboard

1. Purpose of the Document

This document outlines the requirements for a Rate Limiter Dashboard that allows system administrators to monitor and manage API usage, implement rate-limiting policies, and gain real-time insights into traffic patterns. The tool should help ensure fair usage of APIs, prevent abuse, and maintain application performance.

2. Project Overview

The Rate Limiter Dashboard is designed to track and manage API requests per user or IP address, with configurable limits to ensure optimal usage and security.

Key Objectives:

- Monitor API usage in real-time.
- Prevent API abuse by applying rate-limiting policies.
- Provide insights into usage patterns for optimization.

3. Scope

The dashboard will include the following features:

Core Features:

1. Real-Time Monitoring: View live traffic data and API request counts.
2. Rate-Limiting Policies: Set custom limits for different APIs or users.
3. Alert System: Notify administrators when limits are breached.
4. Usage Reports: Generate detailed reports of API usage.
5. Throttling: Temporarily block or reduce API access for excessive requests.

Optional Features:

- Geo-location-based rate limiting.

- Multi-tenancy support for different client organizations.

- Integration with external analytics tools.

4. Functional Requirements

ID	Requirement	Priority
FR1	Track API requests in real-time with Redis.	High
FR2	Apply rate-limiting rules to users or IPs.	High
FR3	Allow administrators to configure rate limits.	High
FR4	Display historical usage data for analysis.	Medium
FR5	Generate and export usage reports in CSV/JSON.	Medium
FR6	Send email/SMS alerts on rate limit breaches.	Medium

5. Non-Functional Requirements

ID	Requirement	Priority
NFR1	System should process 10,000 API requests per second.	High
NFR2	Ensure sub-50ms response time for rate-limiting checks.	High
NFR3	Dashboard should load within 2 seconds.	Medium
NFR4	Ensure secure storage of sensitive configuration data.	Medium
NFR5	Provide robust logging and error tracking.	Medium

6. Database Design

The Rate Limiter Dashboard will use Redis as the primary database for fast in-memory data

processing. Data structures will be designed as follows:

1. **Key Structures:**
   - **String:** To store request counters per user or IP.
   - **Hashes:** For storing user-specific rate-limiting policies.
   - **Sorted Sets:** To track timestamps of requests for precise rate limiting.

2. **Example Schema:**
   - `user:{user_id}:requests`: Counter for API requests.
   - `api:{api_id}:rate_limits`: Rate-limiting configurations.
   - `alerts`: List of triggered alerts for breaches.

3. **Retention Policy:**
   - Use `EXPIRE` for time-based counters to clear old data automatically.

7. Flow Design

The flow of the Rate Limiter Dashboard is as follows:

1. **API Request Processing:**
   - Each incoming request is checked against Redis for existing counters.
   - If the limit is exceeded, the request is rejected, and an alert is logged.

2. **Monitoring:**
   - Real-time metrics are pulled from Redis and displayed on the dashboard.
   - Administrators can view detailed usage graphs.

3. **Alert Handling:**

- Threshold breaches trigger notifications to administrators.
- Alerts are stored in Redis for tracking.

4. **Configuration Updates:**
   - Admin changes to rate-limiting policies are pushed to Redis immediately.

8. Technology Stack

- **Frontend:** React with Chart.js for graphs and TailwindCSS for styling.
- **Backend:** Node.js with Express.js.
- **Database:** Redis for high-speed data handling.
- **Other Tools:**
  - WebSocket for real-time dashboard updates.
  - Docker for containerization.
  - Redis Streams for logging and alert management.

9. Success Criteria

1. The system handles 10,000 requests per second without performance degradation.
2. Alerts are triggered and displayed within 1 second of a breach.
3. Admins can configure and monitor rate-limiting policies seamlessly.

10. Stakeholders

1. **Product Owner:** Defines the vision and manages priorities.
2. **Developers:** Build and maintain the system.
3. **Testers:** Ensure the application is reliable and error-free.
4. **System Administrators:** Use the dashboard to manage APIs.

11. Timeline

Phase	Duration	Deliverable
Requirement Analysis	1 Week	Finalized BRD and technical docs.
Design & Prototyping	2 Weeks	Wireframes and backend architecture
Development	4 Weeks	Fully functional system.
Testing	1 Week	Bug-free system ready for deployment