PROJECT SYNOPSIS

Distributed Rate Limiter System

INTRODUCTION

In today's digital world, almost every service we use—shopping apps, banking, streaming, travel, or even simple login systems—relies on APIs. These APIs sometimes receive unpredictable traffic. For example, during a sale, thousands of users may hit an endpoint at the same moment. If the system cannot handle this load, it may crash.

To prevent such sudden overload or misuse, the industry uses a mechanism called Rate Limiting. It decides how many requests are allowed within a specific time. Our project, the Distributed Rate Limiter System, acts as a smart traffic controller that ensures APIs remain fast, secure, and fair for everyone.

This system is designed to reduce server crashes, protect against abuse, and ensure stable performance—even when millions of users are accessing the app.

PROBLEM STATEMENT

Here are the major problems the system addresses:

1. API Overload
   Too many requests at once slow down servers or cause downtime.
2. Misuse or Attacks
   Bots or malicious users may spam APIs, leading to failures.
3. Unfair Usage
   Some users may overuse free plans unless limits are enforced.
4. Cost Increase
   High API usage increases cloud billing unexpectedly.
5. Distributed System Complexity
   When apps run on multiple servers, enforcing consistent rate limits becomes difficult.

OBJECTIVES

The key goals of our system are:

- To build a fast and reliable rate limiting engine.
- To support multiple strategies like Token Bucket, Sliding Window, and Fixed Window.
- To offer an admin dashboard with real-time monitoring.
- To ensure consistent rate limits across distributed servers.
- To make configuration easy for developers and organizations.

WORKFLOW OF THE SYSTEM

The workflow of the Distributed Rate Limiter System is simple, smooth, and close to how modern API gateways work:

1. Incoming Request

A user sends a request to an application (e.g., login, fetch items, etc.).

2. Application Calls Rate Limiter

Before processing the request, the main application sends the user's key/IP to our Rate Limiter Service through an API call.

3. Check Rate Limits (Core Logic)

The Rate Limiter checks:

- How many requests the user has already made.
- Whether the user exceeds the limit.
- What algorithm is applied (Token Bucket, Sliding Window, etc.).

4. Redis-Based Decision

Since this is a Distributed system, Redis stores counters/tokens centrally:

- Redis increments counters
- Redis checks expiration
- Redis ensures correct limits across all servers

5. Allow or Deny

Based on Redis results:

- If within limit → system returns **ALLOW**
- If exceeded → system returns **DENY**

6. Application Processes or Blocks

- If allowed → request reaches the main service (like ordering food or viewing items).
- If denied → user gets a "Too Many Requests" message.

7. Logging and Analytics

Every decision is logged for:

- Monitoring
- Charts
- Alerts
- Usage reports

8. Dashboard Display

Admins can view:

- Live traffic
- Violations
- High-load endpoints
- API keys
- Custom limit configurations

This workflow ensures system stability without slowing down applications.

SCOPE OF THE PROJECT

The project covers:

- Distributed rate limiting algorithms
- Centralized rule configuration
- Live monitoring dashboard
- API-based integration
- Logging and analytics

Areas not covered:

- Full authentication system
- Payment or subscription management
- AI-based anomaly detection (future extension)

TECHNOLOGY STACK

- Java 17 / Spring Boot
- Redis for distributed counters
- PostgreSQL for configurations
- HTML/CSS/JS + Thymeleaf for dashboard
- Docker for deployment
- Prometheus + Grafana for monitoring

CONCLUSION

- The Distributed Rate Limiter System is a modern solution designed for today's high-traffic applications. It protects APIs, ensures fair usage, prevents crashes, and gives organizations complete control over incoming traffic.
- With its clean architecture, distributed consistency, and easy-to-use dashboard, it offers a highly practical and industry-ready tool that can be integrated into any modern backend.
- Its workflow is simple, intuitive, and powerful—making it not just a project, but a strong demonstration of backend engineering skills.