Coding & Development / API Development

Write code for rate limiting API requests by IP or user, with headers and retry-after logic.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Rate Limiting, API Protection, Security
Updated: May 2026
Why This Prompt Exists
Most APIs have no rate limiting — making them vulnerable to abuse and DoS attacks.

You get:

- no protection against brute force attacks
- API abuse by malicious clients
- server overload from high traffic
- uneven resource consumption
- no way to enforce fair usage

But rate limiting is not optional for public APIs.

It is essential for security and stability.

- Identify client: IP address, user ID, API key
- Storage: in-memory (Redis), database, or file
- Algorithm: fixed window, sliding window, token bucket, leaky bucket
- Response: 429 Too Many Requests with Retry-After header
- Headers: X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset

Without rate limiting, your API is at risk.

This framework forces AI to implement rate limiting.

The Prompt

```
Assume the role of a security engineer who implements API rate
limiting.

Your task is to generate rate limiting middleware code.

Generate:

1. STORAGE SETUP
   - Redis, in-memory map, or database

2. IDENTIFIER EXTRACTION
   - IP address, user ID, or API key

3. RATE LIMIT ALGORITHM
   - Fixed window, sliding window, or token bucket
   - Implementation logic

4. MIDDLEWARE FUNCTION
   - Check current count against limit
   - Increment counter
   - Return 429 if limit exceeded

5. RESPONSE HEADERS
   - X-RateLimit-Limit
   - X-RateLimit-Remaining
   - X-RateLimit-Reset
   - Retry-After
```

6. CONFIGURATION
   - Requests per window
   - Window duration (seconds, minutes, hours)
   - Different limits for different endpoints

INPUTS:

Framework:
[EXPRESS / FASTAPI / DJANGO / OTHER]

Storage Type:
[REDIS / IN-MEMORY / DATABASE / EXTERNAL]

Identify By:
[IP ADDRESS / USER ID / API KEY / COMBINATION]

Rate Limit Values:
[E.G., "100 requests per minute" / "1000 per hour"]

Endpoints to Protect:
[ALL / SPECIFIC (LIST)]

RULES:
- Use Redis for production (in-memory doesn't scale across processes)
- Return 429 Too Many Requests with Retry-After header
- Include rate limit headers in responses (inform clients)
- Different limits for different endpoints (auth endpoints need stricter limits)
- Log rate limit violations for monitoring

```
- Use atomic operations to avoid race conditions
- Make limits configurable (environment variables)
```

How To Use It

- Use Redis for production (in-memory doesn’t scale across multiple server processes).
- Return 429 Too Many Requests with Retry-After header.
- Include rate limit headers in responses so clients can self-regulate.
- Set stricter limits for authentication endpoints (prevent brute force).
- Log rate limit violations for monitoring and abuse detection.
- Use atomic operations (Redis INCR) to avoid race conditions.

Example Input

Framework: EXPRESS

Storage Type: REDIS

Identify By: IP ADDRESS

Rate Limit Values: 100 requests per minute for general endpoints, 10 per minute for auth endpoints

Endpoints to Protect: ALL

Why It Works
Most APIs lack rate limiting.

This framework improves outcomes by forcing:

- storage selection (scalability)
- identifier extraction (client tracking)
- algorithm choice (accuracy)

- response headers (client feedback)
- configurable limits (flexibility)

Great rate limiting doesn’t just block — it informs clients and protects your infrastructure.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, API development frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The OpenAPI/Swagger Generator