

Le rate limiting

Une API sans limite finit toujours par recevoir trop de requêtes : un script en boucle infinie, un client trop enthousiaste, un concurrent qui aspire vos données, ou une attaque par déni de service. Le **rate limiting** limite le nombre de requêtes par client et par période.

Exemple de réponse d'un barman débordé : « reviens dans 10 s ».

Pourquoi limiter ?

- **Disponibilité** : un seul client ne doit pas pouvoir monopoliser le serveur.
- **Coûts** : chaque requête coûte (calcul, base de données, appels à des services payants comme un LLM).
- **Sécurité** : ralentir les attaques par force brute sur une route de connexion.
- **Modèle économique** : offre gratuite à 60 requêtes par heure, offre payante à 5 000 (c'est le cas de l'API GitHub).

Les algorithmes

Algorithme	Principe
Fenêtre fixe	100 requêtes par minute, compteur remis à zéro à chaque minute. Simple, mais permet 200 requêtes en 2 secondes à cheval sur deux minutes.
Fenêtre glissante	100 requêtes sur les 60 dernières secondes, à tout instant. Plus juste.
Seau à jetons (<i>token bucket</i>)	Le client a un seau de jetons qui se remplit à vitesse constante. Chaque requête consomme un jeton. Autorise de petites rafales.

Ce que l'API doit répondre

```
HTTP/1.1 429 Too Many Requests
Retry-After: 10
Content-Type: application/problem+json
```

```
{"title": "Too Many Requests", "status": 429, "detail": "Le barman est débordé, reviens dans 10 s"}
```

Beaucoup d'API informent aussi le client **avant** qu'il atteigne la limite :

```
curl -i https://api.github.com/users/torvalds
```

```
x-ratelimit-limit: 60
x-ratelimit-remaining: 57
x-ratelimit-reset: 1794644000
```

Les headers `X-RateLimit-*` sont une convention répandue mais non standard. L'IETF travaille à un standard (`RateLimit` et `RateLimit-Policy`), déjà supporté par certaines bibliothèques.

Le mettre en place

```
# pip install slowapi
from fastapi import Request
from slowapi import Limiter
from slowapi.errors import RateLimitExceeded
from slowapi.util import get_remote_address

limiter = Limiter(key_func=get_remote_address)
app.state.limiter = limiter

@app.exception_handler(RateLimitExceeded)
async def on_rate_limit(request: Request, exc: RateLimitExceeded):
    response = problem(429, "Too Many Requests", "Le barman est débordé, reviens dans une minute.")
    response.headers["Retry-After"] = "60"
    return response

@app.get("/v1/cocktails")
@limiter.limit("60/minute")
def list_cocktails(request: Request):    # le paramètre request est obligatoire
    ...
```

Limiter par adresse IP est un début, mais tous les étudiants d'une même salle partagent souvent la même IP publique. Une API authentifiée limite plutôt **par clé** ou par utilisateur.

En production, le rate limiting est souvent assuré en amont de l'application : par le reverse proxy (Nginx, Traefik), une passerelle d'API ou un CDN.

Revision #4

Created 2 October 2026 20:00:56 by Thibaud FRICHET

Updated 15 October 2026 09:11:13 by Thibaud FRICHET