

Lidando com rate limiting

Contexto

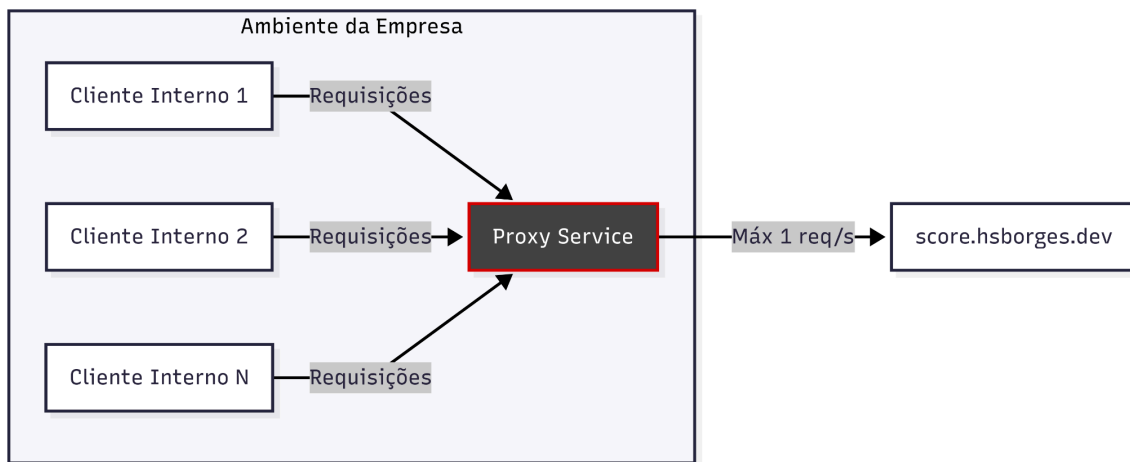
Sua equipe foi encarregada de criar um **serviço de proxy interno** para consumir a API pública de uma empresa parceira disponível em <https://score.hsborges.dev/docs>.

A API impõe **limite de 1 requisição por segundo**. Se o limite for excedido, o provedor aplica uma **penalidade**: a próxima resposta só é liberada **após 2 segundos adicionais** (além da latência normal), impactando todos os serviços internos.

O objetivo é **projetar e implementar** um **proxy resiliente** que:

1. **Acomode picos de requisições internas** sem violar o *rate limit* externo;
2. **Minimize latência média e penalidades**;
3. **Monitore e exponha métricas** para tomada de decisão.

A seguir está um diagrama de alto nível exemplificando o desafio:



Regras do Jogo (restrições)

- **Rate limit** externo fixo: 1 req/s.

- **Penalidade:** ao violar o limite, o provedor impõe **+2s** de atraso.
 - **Ambiente interno ruidoso:** múltiplos clientes internos podem disparar múltiplas requisições simultâneas.
 - **Sem “comprar” mais cota:** o time **não pode** negociar limites maiores — apenas **projetar** para respeitar o limite existente.
 - **Implementação livre:** linguagem, libs e tecnologias à escolha do grupo (documentar).
-

Missão

Projetar e entregar um **serviço proxy** com:

1. **Fila/Buffer interno** para lidar com picos (backpressure);
 2. **Scheduler** para emitir no **máximo 1 chamada/segundo** ao upstream;
 3. **Política de enfileiramento** configurável (ex.: FIFO, prioridade por tipo de operação, deadline/TTL);
 4. **Estratégia de degradação** (fallback) quando a fila cresce demais (ex.: shed load, respostas cacheadas, etc);
 5. **Observabilidade:** logs estruturados, métricas (contadores, histograma de latência, taxa de erro, tamanho da fila), e dashboard simples;
 6. **Configurações** (via arquivo/env): limites, tamanhos de fila, timeouts, política de retry;
-

Requisitos Funcionais

- **RF1:** Expor **GET /proxy/score** que aceita parâmetros da chamada para o upstream.
- **RF2:** Expor **GET /metrics** com métricas de uso
- **RF3:** Expor **GET /health** (liveness/readiness).

Requisitos Não Funcionais

- **RNF1:** Tolerar **rajadas** internas (e.g., 20 req em 1s) sem violar 1 req/s externo.
 - **RNF2: Evitar** penalidades recorrentes. Caso ocorram, **registrar e ajustar** a cadência automaticamente (controle adaptativo).
 - **RNF3: Throughput** sustentado: próximo a 1 req/s para o upstream (estável).
 - **RNF4: Tempo de espera** previsível (fila controlada com limites e políticas de descarte definidas).
 - **RNF5: Observabilidade:** logs/metrics suficientes para auditar decisões do rate limiter e do circuito.
-

Cenários de Teste (aceitação)

Implemente um **test harness** (script ou testes automatizados) que dispare requisições internas e verifique:

1. Rajada controlada

- Enviar 20 requisições em 1 segundo para o proxy.
- **Esperado:** upstream recebe ~1 req/s; zero ou mínimas penalidades; fila cresce, mas estabiliza; nenhuma violação do limite externo.

2. Penalidade proposital

- Simular ou forçar 2+ chamadas paralelas diretas ao upstream (sem o proxy) para observar a penalidade.
- **Esperado:** com o proxy ativo, a **mesma** carga **não** dispara penalidade; métricas mostram "penalties_avoided".

3. Timeout e Circuit Breaker

- Simular que a API remota começa a responder lentamente (> 3s) ou com erros 5xx.
- **Esperado:** o **circuit breaker** abre após limiar, evita novas tentativas por janela configurada, fornece fallback (mensagem amigável/cached), e fecha posteriormente.

4. Política de Fila

- Configurar prioridades por tipo de requisição (p.ex., consultas vs. atualizações).
- **Esperado**: pedidos prioritários passam à frente quando a fila está cheia; pedidos com TTL vencido são descartados (DROPPED) com motivo.

5. Observabilidade

- Métricas refletem: taxa de enfileiramento, tamanho máximo da fila, latência percentil, retries, status do circuito, contagem de quedas por política.

Entregáveis

1. **Código-fonte** (repo público) com README contendo:
 - Arquitetura (diagrama simples), decisões de design e padrões utilizados;
 - Como rodar (Docker Compose ou equivalente), variáveis de ambiente, seed de testes;
 - Endpoints internos e exemplos de uso (curl/HTTPie/Postman/Insomnia).
2. **Relatório técnico curto** (4–6 páginas) contendo:
 - **Padrões** adotados e rejeitados (com justificativa);
 - **Experimentos** de carga (metodologia, cenários, métricas, gráficos);
 - **Análise crítica**: trade-offs (latência x throughput x justiça da fila x experiência do cliente).
3. **Dashboard** mínimo exibindo as métricas chave.
4. **Test Harness** reprodutível (script ou testes automatizados) e registros (logs/prints) dos cenários.

Critérios de Avaliação (rubrica)

- **Aderência ao Problema (20%):** respeita 1 req/s e reduz penalidades?
 - **Qualidade de Arquitetura (25%):** uso consistente de padrões (Proxy, Strategy, Rate Limiter, Circuit Breaker, Command, etc.), isolamento, backpressure.
 - **Confiabilidade & Observabilidade (20%):** métricas, logs, healthchecks, testes de falha.
 - **Desempenho sob Carga (20%):** resultados dos cenários (fila estável, latência previsível, queda controlada).
 - **Clareza de Documentação (15%):** README, ADRs, relatório, exemplos de execução.
-

Segurança & Ética de Uso

- Evitar causar carga desnecessária ao provedor: priorize **simulação** e **limites** claros durante os testes.
 - Nunca expor credenciais em repositórios públicos.
-

Roteiro (Em construção)

1. **Aula 1 — Descoberta & Design:** entender API/contratos, definir métricas, desenhar arquitetura e padrões (esboço).
2. **Aula 2 — MVP:** fila, rate limiter, emissor a 1 req/s, métricas básicas e dois cenários de teste.
3. **Bonus 1 — Resiliência:** circuit breaker, retry com jitter, política de fila/TTL.
4. **Bonus 2 — Observabilidade & Relatório:** dashboard, gráficos, análise crítica.
5. **Demo Day:** apresentação de arquitetura, execução dos testes, discussão de trade-offs.