ESCOLA SESI de refêrencia

Atividade 12 - Arquitetura, Qualidade e Governança de APIs

Thiago Thierry Goudard
Rafael Sonni Bizatto
Gabriel Vieira Alves

Vinicius Tavares Nunes

Síntese Executiva

APIs modernas são essenciais para aplicações conectadas, garantindo comunicação segura, confiável e observável. Este dossiê analisa a GitHub REST API e a Spotify API (incluindo GraphQL), abordando desenho, segurança, observabilidade, governança, qualidade e integração com PHP. Comparações entre REST e GraphQL evidenciam trade-offs em latência, granularidade e versionamento. Aspectos de caching (ETags), webhooks e eventos demonstram desafios de performance e confiabilidade. O estudo aprofunda contratos, evolução de endpoints, autenticação (OAuth2/JWT com escopo mínimo), SLOs, logging estruturado e mitigação de riscos de PII. As decisões arquiteturais são tomadas considerando requisitos não funcionais, fornecendo recomendações para integração robusta, governança eficaz e experiência do desenvolvedor (DX).

3. Introdução

O crescente papel das APIs (Interfaces de Programação de Aplicações) como espinha dorsal de sistemas distribuídos, incluindo aplicações PHP, exige abordagem rigorosa sobre seu ciclo de vida. O objetivo é analisar a segurança, governança e arquitetura de duas APIs públicas maduras — GitHub REST API e Spotify API — para extrair lições arquiteturais aplicáveis.

4. Estilos de API e Trade-offs Arquiteturais

Estilo	Características-chave	Granularidade	Acoplamento	Quando usar
REST	Simples, cache-friendly	Média	Rígido (cliente acoplado)	Integração simples, alto caching
GraphQL	Flexível, evita overfetching	Fina	Flexível	Dashboards e agregação complexa
gRPC	Alto desempenho, HTTP/2	—	Rígido via stubs	Comunicação interna M2M (microservices)

Análise Crítica: REST facilita caching, mas pode gerar overfetching. GraphQL resolve granularidade, útil para dashboards agregados. gRPC é rejeitado para consumo público, indicado para alta performance interna.

5. Contratos e Evolução

- Versionamento: URI, headers ou media type (GitHub e Spotify usam headers + URI).
- Compatibilidade retroativa e depreciação: Mantida para evitar quebra de clientes; comunicada via changelogs e revisão técnica.

- Paginação: GitHub: server-driven (page/limit); Spotify: cursor-based, melhor para dados dinâmicos.

- Idempotência: Garante segurança em múltiplas execuções, essencial para retries confiáveis.

6. Segurança: OAuth2, Escopo Mínimo e Mitigação de Riscos

- Autenticação: OAuth2/JWT; Spotify mais sensível ao escopo que GitHub.

- Proteção de Tokens: Rotação de tokens, TLS obrigatório e mitigação de replay attacks.

- Mitigação de riscos: Rate limiting e circuit breakers protegem contra abuso e falhas.

- Compliance (PII): Minimização de dados e proteção contra vazamentos.

7. Qualidade e Confiabilidade

- Tratamento de erros: Taxonomia clara (HTTP 4xx/5xx). Spotify combina códigos + body detalhado.

- Resiliência: Timeouts e retry/backoff para chamadas críticas; Trace IDs para rastreabilidade.

8. Observabilidade: Monitoramento com SLOs

- Métricas e SLOs: Latência p50/p95/p99, taxa de erro 5xx, saturação de recursos.

- Logging estruturado e tracing distribuído: Fundamental para auditoria e análise de falhas.

9. Desempenho e Cache

- Caching inteligente: ETags e conditional requests reduzem latência e custos.
- Controle de cache: cache-control e surrogate keys em API Gateways.

- Paginação: Cursor-based (Spotify) melhora consistência; server-driven (GitHub) mais simples.

10. Webhooks e Eventos

- GitHub: Pull_request, push, issues; HMAC, retries e DLQs garantem entrega confiável.

- Spotify: Eventos de playlist e follow; limitado e event-driven.

11. Governança e Experiência do Desenvolvedor (DX)

- API Gateways centralizam segurança e rate limiting.

- Revisão técnica obrigatória para novos endpoints.

- Documentação viva via OpenAPI e changelogs claros.

12. Integração com PHP

- Clients resilientes: Guzzle/cURL com retries, backoff e timeouts. ●

Proteção de segredos: Armazenamento seguro de tokens e secrets.

13. Compliance e Custos

- LGPD/PII: Minimização de dados, retenção necessária para auditoria.

- Custos e limites: Quotas (Spotify) e rate limits (GitHub); fallback e graceful degradation.

14. Estudo Comparativo Prático

Aspecto	GitHub REST API	Spotify API	Observações / Justificativa
Autenticação	OAuth2 / PAT	OAuth2 / JWT	Spotify mais sensível ao escopo (segurança)
Paginação	page/limit	cursor-based	Cursor evita inconsistência em dados dinâmicos
Erros	HTTP codes simples	HTTP + body detalhado	Melhor rastreabilidade no Spotify
Webhooks	Push, PR, Issues	Playlist / Follows	GitHub mais robusto (HMAC, retries)

Decisões Arquiteturais Chave:

1. Cursor-based pagination (Spotify): Estabilidade e consistência de dados dinâmicos.
2. ETag + Conditional Requests (Ambas): Redução de latência e custo de largura de banda.
3. Rejeitada: Endpoints sem autenticação para dados sensíveis (Spotify), mitigando riscos de PII.

15. Conclusão e Recomendações

1. Priorizar GraphQL ou cursor-based para dashboards e dados agregados.
2. Implementar retries idempotentes e backoff em clients PHP.
3. Logging estruturado + Trace IDs para monitoramento e auditoria.
4. Aplicar OAuth2/JWT com escopo mínimo para dados sensíveis.
5. Uso inteligente de ETags e conditional requests para otimizar latência e custo.

16. Apêndice Textual: Esqueleto OpenAPI / Contrato

GitHub API – Endpoint de Repositório

- Endpoint: /repos/{owner}/{repo}

- Método: GET
- Autenticação: Header Authorization: token {PAT} (OAuth)

- Parâmetros: owner (string), repo (string)

- Respostas:

    - 200 OK: sucesso, ETag suportado

    - 404 Not Found

    - 500 Internal Error

Spotify API – Endpoint de Playlists

- Endpoint: /users/{user_id}/playlists

- Métodos: GET (Listar), POST (Criar), DELETE (Deletar) ●

  Autenticação: Header Authorization: Bearer {JWT} (OAuth2) ●

  Parâmetros: limit, offset, cursor

- Respostas:

    - 200 OK: sucesso

    - 401 Unauthorized

    - 429 Too Many Requests

- Funcionalidade opcional: Webhook para eventos de playlists