**Relatório Final – GraphQL vs REST: GitHub API**

**Grupo 01:**

Nataniel Geraldo Mendes Peixoto

Nelson de Campos Nolasco

Rubia Coelho de Matos

**1. Introdução**

Este experimento compara o desempenho de chamadas à **GitHub REST API** e à **GitHub GraphQL API**, medindo:

* **RQ1:** Qual API retorna dados mais rapidamente?
* **RQ2:** Qual API devolve payloads menores?

**2. Desenho do Experimento**

**A. Hipóteses**

* **H0:** Não há diferença significativa entre REST e GraphQL (tempo e tamanho).
* **H1:** GraphQL é mais eficiente (menor tempo e tamanho).

**B. Variáveis**

* **Dependentes:** Tempo de resposta (s), Tamanho do payload (bytes)
* **Independente:** Tipo de API (*REST* vs *GraphQL*)

**C. Endpoints**

* **REST API:**
* GET https://api.github.com/repos/{owner}/{repo}
* **GraphQL API:**
* POST https://api.github.com/graphql
* Content-Type: application/json
* {
* "query": "
* query($owner:String!, $repo:String!) {
* repository(owner:$owner, name:$repo) {
* issues(first:100) { totalCount }
* }
* }
* ",
* "variables": { "owner": "{owner}", "repo": "{repo}" }
* }
* ``` :contentReference[oaicite:3]{index=3}

**D. Procedimento**

1. Configurar GITHUB\_TOKEN e parâmetros owner, repo e trials.
2. Executar experiment.py, que grava as medições em experiment\_results.csv.
3. Rodar analysis.py para gerar experiment\_summary.csv.
4. Gerar histogramas com dashboard.py.

**3. Resultados**

**3.1 Estatísticas Agregadas**

A seguir, as médias e desvios padrões de tempo de resposta e tamanho de payload, agrupadas por tipo de API:

| **API** | **Tempo Médio (s)** | **Desv. Padrão Tempo (s)** | **Tamanho Médio (bytes)** | **Desv. Padrão Tamanho (bytes)** |
| --- | --- | --- | --- | --- |
| **REST** | 0.3289 | 0.0377 | 6 719 | 0 |
| **GraphQL** | 0.3461 | 0.0557 | 51 | 0 |

Esses valores foram extraídos do arquivo experiment\_summary.csv após 30 medições para cada API.

**3.2 Observações Específicas**

* **REST** apresentou o menor tempo médio de resposta (0.3289 s) e alta consistência (std = 0.0377 s).
* **GraphQL** teve tempo médio ligeiramente maior (0.3461 s) e variação um pouco maior (std = 0.0557 s).
* O payload da **GraphQL** é drasticamente menor (51 bytes), enquanto o da **REST** inclui um objeto de repositório completo (~6.7 KB).

**4. Discussão**

* **Desempenho de Latência (RQ1):** Contrariando a hipótese (H1), a REST API mostrou-se um pouco mais rápida, provavelmente devido à simplicidade da chamada GET ao endpoint de repositório, enquanto o POST GraphQL exige parsing extra no servidor e payload de requisição maior.
* **Tamanho do Payload (RQ2):** Conforme esperado, a GraphQL retorna payloads muito menores, uma vez que foi projetada para retornar apenas os campos solicitados, reduzindo transferência de dados e potencialmente melhorando performance em conexões lentas.
* **Limitações:**
  + Rede local e efeitos de cache do GitHub podem ter influenciado a latência.
  + Apenas um endpoint (repositório + issues) foi testado; outros cenários podem se comportar de forma distinta.
  + O número de trials (30) é razoável, mas mais rodadas poderiam reduzir incertezas estatísticas.

**5. Conclusões**

1. **Velocidade:** Para chamadas simples de leitura de metadados de repositório, a REST API mostrou latência ligeiramente inferior.
2. **Eficiência de Dados:** A GraphQL API entrega payloads muito menores, economizando largura de banda, o que pode ser crucial em ambientes com restrição de dados.
3. **Recomendação Prática:** Se a prioridade for **latência** e simplicidade de uso, escolha a REST API. Se o foco for **otimização de banda** e flexibilidade na seleção de campos, prefira a GraphQL API.
4. **Trabalhos Futuros:**
   * Testar múltiplos endpoints (issues, pull requests, commits) e outros tipos de queries GraphQL mais complexas.
   * Realizar benchmarks em rede real (3G/4G/5G) e com diferentes tamanhos de repositório para validar resultados.