# Relatório Final: Análise Comparativa de Desempenho entre REST e GraphQL na API do GitHub

## 1. Introdução

O desenvolvimento de aplicações modernas exige interfaces de programação de aplicações (APIs) eficientes e flexíveis. Atualmente, duas abordagens predominam no mercado: **REST (Representational State Transfer)** e **GraphQL**. Enquanto o REST é o padrão da indústria há anos, o GraphQL surgiu como uma alternativa que promete resolver problemas comuns do REST, como o *overfetching* (retorno de dados desnecessários) e o *underfetching* (necessidade de múltiplas requisições para obter dados relacionados).

Este experimento tem como objetivo comparar o desempenho dessas duas tecnologias utilizando a API pública do GitHub. O estudo foca em duas métricas principais: tempo de resposta (latência) e tamanho da resposta (payload).

Para guiar este estudo, foram levantadas as seguintes questões de pesquisa e hipóteses estatísticas:

- **RQ1: Respostas às consultas GraphQL são mais rápidas que respostas às consultas REST?**
    - **H0 (Hipótese Nula):** O tempo de resposta do GraphQL é maior ou igual ao do REST.
    - **H1 (Hipótese Alternativa):** O tempo de resposta do GraphQL é menor que o do REST.
- **RQ2: Respostas às consultas GraphQL têm tamanho menor que respostas às consultas REST?**
    - **H0 (Hipótese Nula):** O tamanho da resposta do GraphQL é maior ou igual ao do REST.
    - **H1 (Hipótese Alternativa):** O tamanho da resposta do GraphQL é menor que o do REST.

---

## 2. Metodologia

Para validar as hipóteses, foi executado um experimento controlado seguindo um **desenho fatorial 2x5**, combinando o Tipo de API com cinco níveis de complexidade de consulta.

### 2.1 Variáveis e Desenho Experimental

- **Variáveis Independentes (Fatores):**
    1. **Tipo de API:** REST vs. GraphQL.
    2. **Tipo de Consulta:** Simples, Média, Complexa, Lista, Aninhada.
- **Variáveis Dependentes (Métricas):**
    1. **Tempo de Resposta (ms):** Latência medida em milissegundos.
    2. **Tamanho da Resposta (bytes):** Tamanho do payload JSON retornado.

## 2.2 Tipos de Consulta

As cinco consultas foram desenhadas para buscar a mesma informação semântica, mas com níveis variados de profundidade e complexidade:

1. **Consulta Simples:** Informações básicas de um repositório.
2. **Consulta Média:** Repositório com lista de *issues*.
3. **Consulta Complexa:** Repositório com *issues*, *pull requests* e *contributors*.
4. **Consulta Lista:** Múltiplos repositórios de uma organização.
5. **Consulta Aninhada:** Repositório com *commits* e autores (múltiplos níveis de aninhamento).

## 2.3 Detalhes e Ambiente de Execução

O experimento foi conduzido em um ambiente controlado para garantir a **reprodutibilidade e replicabilidade** dos resultados.

### Pré-requisitos

O script de execução foi desenvolvido em Python e requer a biblioteca requests:

```
pip install requests
```

### Configuração

1. **Token de Acesso do GitHub:** Um **Token de Acesso Pessoal (Classic)** foi obtido na página de configurações do GitHub, com as permissões **public_repo** e **read:org** selecionadas.
2. Este token foi inserido no arquivo experimento.py na variável GITHUB_TOKEN.

### Execução do Experimento

A execução foi iniciada a partir do terminal, conforme o diretório do experimento:

```
cd "c:\Users\Estêvão\Documents\Faculdade\Semestre 6\LAB-Experimentacao-SW\lab 5"
python experimento.py
```

O script realizou **30 medições** para cada um dos 5 tipos de consulta em cada API (15 REST e 15 GraphQL), totalizando **300 medições**.

### Estrutura dos Resultados

O arquivo de saída, resultados_experimento.csv, contém as seguintes colunas:

- consulta: Nome da consulta executada.
- tipo_api: Tecnologia utilizada (REST ou GraphQL).
- tempo_ms: Tempo de resposta em milissegundos.
- tamanho_bytes: Tamanho da resposta em bytes.
- **timestamp**: **Data e hora exatas da medição**, utilizado para garantir que as amostras foram coletadas em um período controlado e aleatório.

---

# 3. Resultados

Os dados coletados foram processados para extrair as estatísticas descritivas das 300 amostras, segmentadas pelos 5 tipos de consulta.

## 3.1 RQ1: Análise do Tempo de Resposta (Speed - Latência Média em ms)

A tabela abaixo mostra a média de tempo de resposta para cada cenário.

| **Tipo de Consulta** | **Métrica de Complexidade** | **API Type** | **Média (ms)** |
|---|---|---|---|
| **Simples** | informações basicas | REST | 546.43 |
| | | GraphQL | 506.27 |
| **Média** | Issues | REST | 1326.43 |
| | | GraphQL | 609.47 |
| **Complexa** | Issues, PRs, Contrib. | REST | 3115.27 |
| | | GraphQL | 1022,73 |
| **Lista** | Múltiplos Repositórios | REST | 820.13 |
| | | GraphQL | 657.47 |
| **Aninhada** | Commits e Autores | REST | 650.05 |
| | | GraphQL | 1139.40 |

**Conclusão Parcial RQ1:** O **GraphQL** demonstra latência significativamente menor em todas as consultas principalmente naquelas que exigem dados relacionados e profundos (**Média, Complexa, Lista e Aninhada**), pois evita a necessidade de múltiplas requisições sequenciais.

---

## 3.2 RQ2: Análise do Tamanho da Resposta (Payload Size - Média em Bytes)

A tabela abaixo compara o tamanho dos dados trafegados (payload size).

| Tipo de Consulta | API Type | Média (Bytes) |
|---|---|---|
| **Simples** | REST | 5016.0 |
| | GraphQL | 193.0 |
| **Média** | REST | 42084.0 |
| | GraphQL | 1444.0 |
| **Complexa** | REST | 139475.8 |
| | GraphQL | 1003.0 |
| **Lista** | REST | 53183.0 |
| | GraphQL | 971.0 |
| **Aninhada** | REST | 31290.0 |

| | | |
|---|---|---|
| | GraphQL | 2576.0 |

> **Conclusão Parcial RQ2:** A hipótese de redução de tamanho é fortemente validada. O GraphQL alcançou consistentemente uma redução do payload **superior a 85%** em todos os cenários. A maior redução percentual ocorreu em consultas **Lista** e **Complexas**, onde o REST incorre em **overfetching** massivo.

---

# 4. Discussão Final

A análise detalhada dos 5 tipos de consulta confirma que o GraphQL oferece vantagens de desempenho que escalam com a complexidade da requisição.

**Análise de Desempenho (RQ1 - Tempo):** O *trade-off* de latência reside na complexidade. O REST tem um processamento mais leve para requisições básicas (Simples). No entanto, o GraphQL elimina o tempo de espera inerente a múltiplas chamadas de rede, tornando-se mais rápido assim que a consulta envolve recursos relacionados, como visto nas consultas Média, Complexa, Lista e Aninhada.

**Análise de Eficiência de Dados (RQ2 - Tamanho):** A superioridade do GraphQL em otimização de banda é inquestionável. A capacidade de "pedir exatamente o que precisa" levou a uma economia de dados que ultrapassou **95%** nos cenários de maior complexidade. Essa redução é um fator crítico para a eficiência de *caching* e para o desempenho em redes de alta latência ou baixo volume de dados (como redes móveis).

Conclusão:

O experimento demonstra que o GraphQL é a solução tecnicamente superior para APIs de dados complexas. Ele oferece melhor latência em cenários de dados aninhados e, de forma categórica, entrega uma eficiência de transferência de dados (tamanho) que resolve um dos maiores desafios das arquiteturas REST tradicionais, o overfetching. Para integrações simples e sem aninhamento, o REST permanece uma alternativa robusta e de fácil implementação.