Relatório Técnico – LAB 05: REST vs GraphQL – Um experimento controlado

---

1. Informações do grupo
Curso: Engenharia de Software
Disciplina: Laboratório de Experimentação de Software
Período: 6º Período
Professor(a): Prof. Dr. João Paulo Carneiro Aramuni
Membros do Grupo: <Preencher com os nomes dos integrantes>

---

2. Introdução

Este laboratório investiga o impacto do estilo de API (REST vs GraphQL) no desempenho de um sistema de catálogo de jogos. O objetivo é comparar, para um mesmo conjunto de consultas, se há diferença no tempo de resposta e no tamanho do payload entre os dois estilos, quando executados sobre o mesmo backend e dados.

O contexto escolhido foi um catálogo de jogos digitais, com as entidades principais de jogos, usuários e avaliações. Sobre esse domínio, foram definidos três cenários de consulta:

- **Cenário A – Lista de jogos**: listagem paginada de jogos com poucos atributos.
- **Cenário B – Jogo com reviews**: detalhes de um jogo específico e suas avaliações.
- **Cenário C – Usuário com biblioteca**: dados de um usuário e a lista de jogos da sua biblioteca.

Cada cenário é executado tanto em uma API REST quanto em uma API GraphQL, sob as mesmas condições de ambiente, permitindo uma comparação direta entre os dois estilos.

2.1 Questões de Pesquisa (Research Questions – RQs)

As questões de pesquisa foram definidas para guiar a investigação:

RQ | Pergunta
RQ1 | Para as mesmas consultas, as respostas GraphQL são mais rápidas que REST?
RQ2 | Para as mesmas consultas, as respostas GraphQL têm tamanho menor que REST?

2.2 Hipóteses Informais (Informal Hypotheses – IH)

Com base na literatura e nas vantagens normalmente atribuídas ao GraphQL, foram definidas as seguintes hipóteses informais:

IH | Descrição
IH01 | Em cenários mais complexos (com dados relacionados), o GraphQL apresentará tempo de resposta igual ou ligeiramente melhor que o REST.
IH02 | O GraphQL produzirá respostas menores em bytes em pelo menos parte dos cenários, especialmente quando a consulta eliminar campos não utilizados.
IH03 | A diferença de desempenho entre REST e GraphQL variará de acordo com o tipo de cenário (lista simples, dados relacionais ou consulta aninhada).

---

3. Tecnologias e ferramentas utilizadas

- **Linguagem de Programação**: Python 3
- **Frameworks/Bibliotecas Backend**:
    - Flask (implementação dos endpoints REST e exposição do endpoint /graphql)
    - Graphene (definição do schema GraphQL)
- **Script de experimento**: measure_requests.py, utilizando a biblioteca requests para envio das requisições e medição.
- **Dependências principais**: flask, graphene, requests, além de bibliotecas padrão da linguagem (por exemplo, time).
- **Ferramentas de apoio**:
    - Cliente HTTP (Postman ou Insomnia) para testes manuais
    - Editor de código (VS Code ou equivalente)
    - Git e GitHub para versionamento do código

---

4. Metodologia

Nesta seção são descritas as etapas do experimento: definição dos cenários, preparação do ambiente, geração de dados e coleta automatizada das medições.

4.1 Coleta de dados

A coleta foi realizada a partir de um script automatizado em Python (measure_requests.py), que envia requisições para as duas APIs (REST e GraphQL) executando os três cenários de consulta definidos:

- **Cenário A (simples)**:
    - Lista paginada de jogos com poucos atributos (id, name, genre).
- **Cenário B (relacional)**:
    - Um jogo específico com suas avaliações (reviews).
- **Cenário C (aninhado/complexo)**:
    - Um usuário com sua biblioteca de jogos (lista de jogos associados ao usuário).

Para cada cenário (A, B, C) e para cada tipo de API (REST, GraphQL), foram executadas 30 repetições da mesma requisição, totalizando:

- 3 cenários × 2 APIs × 30 repetições = **180 medições de tempo de resposta**
- As mesmas 180 medições de **tamanho de resposta**

O script registra cada chamada em um arquivo CSV (results.csv), contendo as seguintes colunas:

- api_type – tipo de API (rest ou graphql)
- scenario – cenário de consulta (A, B ou C)
- repetition – número da repetição (1 a 30)
- response_time_ms – tempo de resposta em milissegundos
- response_size_bytes – tamanho do corpo da resposta em bytes
- status_code – código de status HTTP retornado

4.2 Filtragem e controle de ambiente

Como o experimento foi executado em ambiente local (máquina do desenvolvedor), algumas medidas foram adotadas para reduzir variações indesejadas:

- Execução do servidor Flask em modo estável, evitando alterações de código durante a coleta.
- Uso da mesma máquina e do mesmo servidor para os dois estilos de API, compartilhando exatamente a mesma estrutura de dados em memória (listas de jogos, usuários e reviews).

- Manutenção da máquina relativamente ociosa durante a coleta, evitando processos pesados em paralelo.
- Número fixo de repetições (30) por combinação API × cenário, permitindo comparações mais robustas.

Não foi realizada filtragem ou remoção de outliers, pois o objetivo do laboratório é observar o comportamento "bruto" das medições em um ambiente controlado, e todos os valores coletados se mostraram consistentes (status 200 em todas as requisições).

4.3 Normalização e pré-processamento

Os dados gerados já são produzidos em formato tabular padronizado pelo script de medição:

- Uma linha por requisição
- Colunas: api_type, scenario, repetition, response_time_ms, response_size_bytes, status_code

Não foi necessária normalização de escalas, pois:

- O tempo já está em milissegundos
- O tamanho já está em bytes

O pré-processamento se limitou à leitura do arquivo CSV em ferramentas de análise (por exemplo, Pandas) e ao agrupamento por cenário e tipo de API para cálculo das estatísticas descritivas.

4.4 Métricas

Neste experimento foram definidas duas métricas principais, diretamente ligadas às RQs:

Código | Métrica | Descrição
LM01 | Tempo de resposta (ms) | Tempo entre o envio da requisição e o recebimento completo da resposta, em milissegundos.
LM02 | Tamanho da resposta (bytes) | Quantidade de bytes do corpo da resposta HTTP (payload), desconsiderando cabeçalhos.

4.5 Cálculo de métricas

As métricas são calculadas diretamente no script de medição:

- **Tempo de resposta (LM01)**:
    - Obtido pela diferença entre time.perf_counter() antes e depois da requisição.
    - O valor em segundos é convertido para milissegundos multiplicando por 1000 e armazenado em response_time_ms.
- **Tamanho da resposta (LM02)**:
    - Calculado via len(resp.content), que retorna o número de bytes do corpo da resposta.
    - Armazenado em response_size_bytes.

Cada medição é registrada como uma linha no CSV com as colunas:
api_type, scenario, repetition, response_time_ms, response_size_bytes, status_code.

4.6 Ordenação e análise inicial

Após a coleta, os dados foram analisados agrupando por cenário e tipo de API. Para cada combinação (por exemplo, cenário A com REST, cenário A com GraphQL etc.), foram calculados:

- Média, mediana, desvio padrão, mínimo e máximo do tempo de resposta (LM01).
- Média, mediana, desvio padrão, mínimo e máximo do tamanho da resposta (LM02).

Essa análise descritiva permitiu:

- Verificar se havia diferenças visíveis entre REST e GraphQL em cada cenário.
- Identificar se alguma combinação apresentava comportamento fora do padrão (o que não ocorreu).

4.7 Relação das RQs com as Métricas

RQ | Pergunta | Métrica utilizada | Código da Métrica
RQ1 | As respostas GraphQL são mais rápidas que as respostas REST? | Tempo de resposta (ms) | LM01
RQ2 | As respostas GraphQL têm tamanho menor do que as respostas REST? | Tamanho da resposta (bytes) | LM02

---

5. Resultados

A seguir são apresentados os resultados obtidos a partir das 180 medições realizadas.

5.1 Distribuição por categoria (API × Cenário)

A Tabela 1 resume, para cada cenário e tipo de API, o tempo médio de resposta e o tamanho médio da resposta.

Tabela 1 – Médias de tempo e tamanho por cenário e tipo de API

Cenário | API | Tempo médio (ms) | Tamanho médio (bytes)
A | REST | 2047,4 | 1367
A | GraphQL | 2045,9 | 1804
B | REST | 2044,2 | 548
B | GraphQL | 2043,1 | 517
C | REST | 2049,7 | 1700
C | GraphQL | 2054,3 | 2152

Observa-se que:

- Os tempos médios estão sempre próximos de 2040–2055 ms, independentemente do cenário ou API.
- Os tamanhos médios das respostas variam bastante entre cenários e, em dois deles (A e C), as respostas GraphQL são significativamente maiores que as respostas REST.

5.2 Estatísticas descritivas

A Tabela 2 apresenta, para cada combinação cenário × API, as estatísticas descritivas do tempo de resposta (LM01) e do tamanho da resposta (LM02).

Tabela 2 – Estatísticas descritivas por cenário e tipo de API

Cenário A – REST (n = 30)

- Tempo médio (ms): 2047,4
- Mediana (ms): 2050,9
- Desvio padrão (ms): 10,4
- Mínimo (ms): 2023,9
- Máximo (ms): 2065,5
- Tamanho médio (bytes): 1367

- Mediana (bytes): 1367
- Mínimo (bytes): 1367
- Máximo (bytes): 1367

Cenário A – GraphQL (n = 30)

- Tempo médio (ms): 2045,9
- Mediana (ms): 2044,7
- Desvio padrão (ms): 8,5
- Mínimo (ms): 2028,7
- Máximo (ms): 2059,7
- Tamanho médio (bytes): 1804
- Mediana (bytes): 1804
- Mínimo (bytes): 1804
- Máximo (bytes): 1804

Cenário B – REST (n = 30)

- Tempo médio (ms): 2044,2
- Mediana (ms): 2044,1
- Desvio padrão (ms): 8,9
- Mínimo (ms): 2021,7
- Máximo (ms): 2063,1
- Tamanho médio (bytes): 548
- Mediana (bytes): 548
- Mínimo (bytes): 548
- Máximo (bytes): 548

Cenário B – GraphQL (n = 30)

- Tempo médio (ms): 2043,1
- Mediana (ms): 2044,7

- Desvio padrão (ms): 8,5
- Mínimo (ms): 2030,4
- Máximo (ms): 2061,4
- Tamanho médio (bytes): 517
- Mediana (bytes): 517
- Mínimo (bytes): 517
- Máximo (bytes): 517

Cenário C – REST (n = 30)

- Tempo médio (ms): 2049,7
- Mediana (ms): 2053,0
- Desvio padrão (ms): 7,8
- Mínimo (ms): 2029,6
- Máximo (ms): 2061,0
- Tamanho médio (bytes): 1700
- Mediana (bytes): 1700
- Mínimo (bytes): 1700
- Máximo (bytes): 1700

Cenário C – GraphQL (n = 30)

- Tempo médio (ms): 2054,3
- Mediana (ms): 2055,8
- Desvio padrão (ms): 12,9
- Mínimo (ms): 2030,2
- Máximo (ms): 2074,8
- Tamanho médio (bytes): 2152
- Mediana (bytes): 2152
- Mínimo (bytes): 2152

- Máximo (bytes): 2152

5.3 Gráficos

- **Figura 1 – Boxplot do tempo de resposta por API e cenário**
    - Eixo X: combinação (A–REST, A–GraphQL, B–REST, etc.)
    - Eixo Y: tempo de resposta (ms)
- **Figura 2 – Gráfico de barras do tamanho médio das respostas por API e cenário**
    - Comparando, em cada cenário, o tamanho médio em bytes de REST vs GraphQL.
- **Figura 3 – Gráfico de linhas do tempo médio por cenário, separado por API**
    - Mostrando que as curvas de REST e GraphQL praticamente se sobrepõem.

5.4 Discussão dos resultados

Nesta seção, os resultados são confrontados com as hipóteses informais definidas na Seção 2.2.

- **RQ1 – Tempo de resposta**
    - As médias de tempo de resposta de REST e GraphQL são praticamente idênticas em todos os cenários (diferenças inferiores a 0,25%).
    - Isso indica que, neste experimento, o estilo de API por si só não produziu vantagem significativa de desempenho em tempo.
    - O custo dominante parece ser o próprio processamento da requisição no servidor Flask, que é compartilhado pelos dois estilos.

→ **IH01** (“GraphQL seria igual ou ligeiramente melhor em cenários complexos”) não foi confirmada de forma clara.
Em alguns cenários o GraphQL é marginalmente mais rápido, em outros é marginalmente mais lento, mas tudo dentro da faixa de ruído de medição.

- **RQ2 – Tamanho da resposta**

- No cenário A (lista de jogos) e no cenário C (usuário com biblioteca), o tamanho médio das respostas GraphQL foi significativamente maior do que o de REST:
    - Cenário A: 1804 bytes (GraphQL) vs 1367 bytes (REST) → ~32% maior.
    - Cenário C: 2152 bytes (GraphQL) vs 1700 bytes (REST) → ~26,6% maior.
- Apenas no cenário B (jogo com reviews) o GraphQL produziu uma resposta ligeiramente menor:
    - 517 bytes (GraphQL) vs 548 bytes (REST) → ~5,7% menor.

➜ Isso mostra que o GraphQL **não garante** payloads menores por padrão.
O tamanho depende muito de como a query é escrita e de quais campos são solicitados. Se a query pede mais campos do que o endpoint REST, a resposta fica maior; se a query é enxuta, pode ficar menor.

➜ **IH02** ("GraphQL produziria respostas menores em pelo menos parte dos cenários") foi parcialmente confirmada: houve apenas um cenário com vantagem em tamanho (B), e dois cenários em que o GraphQL foi claramente pior em termos de bytes.

- **IH03 – Variação por tipo de cenário**
    - De fato, a diferença entre REST e GraphQL varia por cenário:
        - Em B, GraphQL conseguiu ser um pouco mais econômico em bytes.
        - Em A e C, GraphQL devolveu mais dados que REST.
    - Isso reforça que o impacto da tecnologia depende do desenho da consulta e da estrutura dos dados, e não é uma propriedade fixa da ferramenta.

➜ **IH03** é confirmada: a diferença entre REST e GraphQL não é uniforme, mas dependente do tipo de cenário.

Em resumo: o experimento não confirma a narrativa de que "GraphQL é sempre mais rápido e sempre devolve menos dados". Ele mostra um cenário mais realista: GraphQL dá flexibilidade, mas essa flexibilidade precisa ser usada com cuidado; caso contrário, você pode acabar mandando mais dados do que um endpoint REST bem projetado, sem ganho perceptível de tempo.

---

6. Conclusão

Este laboratório implementou e avaliou um experimento controlado comparando REST e GraphQL em um sistema de catálogo de jogos, com três cenários de consulta (lista de jogos, jogo com reviews e usuário com biblioteca). Para cada cenário e estilo de API, foram coletadas 30 medições de tempo e tamanho de resposta, totalizando 180 observações.

Os principais achados foram:

- **Tempo de resposta (RQ1)**:
    - Não foi observada diferença prática de desempenho entre REST e GraphQL.
    - As médias de tempo ficaram na faixa de 2043–2054 ms em todos os cenários, com diferenças inferiores a 0,25% entre as tecnologias.
    - Assim, neste contexto, não há evidência de que GraphQL seja mais rápido que REST.
- **Tamanho da resposta (RQ2)**:
    - O GraphQL não apresentou vantagem consistente em termos de tamanho do payload.
    - Em dois cenários (A e C), o payload GraphQL foi de aproximadamente 25–32% maior que o de REST, e em apenas um cenário (B) foi cerca de 6% menor.
    - Isso reforça a ideia de que o benefício de GraphQL em redução de tráfego não é automático; depende de queries bem desenhadas.

Quanto às hipóteses:

- **IH01** (GraphQL ligeiramente melhor em tempo) não foi confirmada.
- **IH02** (GraphQL com respostas menores em parte dos cenários) foi parcialmente confirmada, com vantagem apenas em um cenário.
- **IH03** (diferença varia por tipo de cenário) foi confirmada, mostrando que o impacto depende do padrão de consulta.

Como limitações, destacam-se:

- Ambiente local, sem concorrência de usuários e sem latências de rede realistas.
- Backend simples em Flask, sem camadas adicionais de cache, banco de dados externo ou balanceamento de carga.

Como trabalho futuro, seria interessante:

- Repetir o experimento em um ambiente mais próximo de produção (por exemplo, em um serviço em nuvem).
- Variar o desenho das queries GraphQL para explorar cenários onde o cliente pede estritamente o mínimo de dados necessário.
- Investigar também métricas de consumo de CPU/memória no servidor e o impacto de múltiplos clientes concorrentes.

---

7. Referências

- Documentação do Flask
- Documentação do Graphene (GraphQL para Python)
- Documentação da biblioteca requests
- Material da disciplina de Laboratório de Experimentação de Software – Prof. João Paulo Carneiro Aramuni

---

8. Apêndices

- **Apêndice A – Código-fonte do servidor (app.py)**
    - Implementação da API REST e do endpoint GraphQL, com geração dos dados em memória.
- **Apêndice B – Script de medição (measure_requests.py)**
    - Script responsável por enviar as requisições, medir o tempo e registrar os resultados em CSV.
- **Apêndice C – Arquivo de resultados (results.csv)**
    - Contém as 180 medições utilizadas neste relatório, com as colunas:
        - api_type, scenario, repetition, response_time_ms, response_size_bytes, status_code.