

LAB 5

1.1 Contexto e objetivo

Este experimento compara o uso de **REST** e **GraphQL** na implementação de uma API para um **catálogo de jogos**.

Ambas as APIs expõem os mesmos dados (jogos, usuários, avaliações), mas com estilos de acesso diferentes.

Objetivo geral:

Avaliar se, para o mesmo conjunto de consultas, há diferença entre REST e GraphQL em termos de:

- **Tempo de resposta** (ms)
- **Tamanho da resposta** (bytes)

1.2 Questões de pesquisa

- **RQ1:** As respostas às consultas GraphQL são mais rápidas que as respostas às consultas REST?
- **RQ2:** As respostas às consultas GraphQL têm tamanho menor do que as respostas às consultas REST?

1.3 Hipóteses

Para cada questão de pesquisa, definimos hipóteses nulas e alternativas.

Para RQ1 (tempo):

- **H0_RQ1 (nula):**
Não há diferença estatisticamente significativa no tempo médio de resposta entre REST e GraphQL para as mesmas consultas.
- **H1_RQ1 (alternativa):**
O tempo médio de resposta das consultas com GraphQL é **menor** do que o das consultas com REST para as mesmas operações.

Para RQ2 (tamanho da resposta):

- **H0_RQ2 (nula):**
Não há diferença estatisticamente significativa no tamanho médio das respostas entre REST e GraphQL para as mesmas consultas.
- **H1_RQ2 (alternativa):**
O tamanho médio das respostas com GraphQL é **menor** do que o das respostas com REST para as mesmas operações.

1.4 Variáveis

Variáveis dependentes (o que é medido):

- **VD1 – Tempo de resposta (ms):**
Tempo total entre o envio da requisição e o recebimento completo da resposta.
- **VD2 – Tamanho da resposta (bytes):**
Quantidade de bytes do corpo da resposta (payload), sem contar cabeçalhos HTTP.

Variáveis independentes (fatores de comparação):

- **VI1 – Tipo de API**
 - Nível 1: REST
 - Nível 2: GraphQL
- **VI2 – Tipo de consulta (cenário de uso)**
Três cenários representativos do catálogo de jogos:
 - **Cenário A (simples):**
Buscar lista paginada de jogos com poucos atributos (id, nome, gênero).
 - **Cenário B (relacional):**
Buscar um jogo específico com suas avaliações e média de nota.
 - **Cenário C (aninhado/complexo):**
Buscar um usuário, suas listas de jogos e detalhes de cada jogo da lista.

1.5 Tratamentos

Cada combinação de API × cenário é um tratamento.

- **T1A:** REST – Cenário A
- **T1B:** REST – Cenário B

- **T1C:** REST – Cenário C
- **T2A:** GraphQL – Cenário A
- **T2B:** GraphQL – Cenário B
- **T2C:** GraphQL – Cenário C

Os tratamentos serão comparados sempre em pares REST vs GraphQL para o mesmo cenário.

1.6 Objetos experimentais

- **OE1 – APIs desenvolvidas:**
 - Uma API REST para o catálogo de jogos.
 - Uma API GraphQL para o mesmo catálogo, utilizando a mesma base de dados em memória.
- **OE2 – Conjunto de consultas:**
 - As consultas A, B e C definidas acima, que representam cenários reais de uso do sistema.

1.7 Tipo de projeto experimental

- **Tipo:** Experimento controlado em ambiente de laboratório.
- **Desenho:** Medidas repetidas / pareadas.
 - A mesma consulta (por exemplo, Cenário A) é executada em REST e em GraphQL sob as mesmas condições de ambiente, permitindo comparar os resultados em pares.

1.8 Quantidade de medições

Para cada **cenário (A, B, C)** e cada **tipo de API (REST, GraphQL)**:

- Serão executadas **30 repetições** da mesma requisição.

Total planejado:

- $3 \text{ cenários} \times 2 \text{ APIs} \times 30 \text{ repetições} = 180 \text{ medições}$ de tempo
- E as mesmas 180 medições de tamanho de resposta.

1.9 Ameaças à validade

Validade interna

- Variação de carga na máquina durante a execução (outros processos influenciando tempo).
- Cache de banco de dados ou da API, tornando algumas execuções artificialmente mais rápidas.
- Ordem fixa (sempre REST antes de GraphQL) pode gerar “efeito de aquecimento”.

Mitigação:

- Minimizar processos paralelos na máquina durante o experimento.
- Executar uma fase de “aquecimento” antes de começar a coletar dados.
- Alternar a ordem dos testes (em parte das execuções começar por REST, em outra por GraphQL).

Validade externa

- O domínio (catálogo de jogos) é relativamente simples e pode não representar sistemas muito grandes ou com lógica de negócio complexa.
- O experimento é realizado em rede local, sem latência de internet real.

Mitigação:

- Descrever claramente o contexto e o tamanho do sistema.
- Deixar explícito que os resultados se aplicam principalmente a sistemas de porte semelhante.

Validade de construto

- Medir apenas tempo e tamanho de resposta não cobre outros aspectos importantes (manutenibilidade, curva de aprendizado, segurança, etc.).

Mitigação:

- Deixar claro que o objetivo do experimento é avaliar **desempenho de resposta e eficiência de payload**, e não todas as dimensões de qualidade de uma API.

Validade de conclusão

- Tamanho de amostra limitado pode diminuir o poder estatístico.
- Outliers extremos podem distorcer médias.

Mitigação:

- Verificar possíveis outliers e decidir, com critério, se serão removidos.
- Utilizar também medidas robustas (mediana) na análise, não apenas médias.

2. Preparação do Experimento:

Nesta etapa foi montado o cenário experimental para aplicação dos tratamentos definidos no Passo 1. As APIs (REST e GraphQL), os scripts de medição e os dados de teste foram organizados em um repositório no GitHub do projeto.

2.1 Ambiente e ferramentas

O backend foi desenvolvido em **Python**, usando **Flask** para expor tanto a API REST quanto o endpoint GraphQL, em conjunto com a biblioteca **Graphene** para definição do schema GraphQL.

As duas APIs utilizam a mesma fonte de dados, composta por estruturas em memória que simulam um catálogo de jogos, usuários e avaliações.

Para medição, é utilizado um script em **Python** com a biblioteca **requests**, responsável por enviar requisições e registrar tempo e tamanho das respostas em arquivos CSV.

Ferramentas como Postman/Insomnia foram usadas apenas para testes manuais durante o desenvolvimento.

2.2 Implementação das APIs

A API REST expõe endpoints específicos para cada cenário:

- Cenário A: GET /rest/games?page=1&limit=20

- Cenário B: GET /rest/games/{id}
- Cenário C: GET /rest/users/{id}/library

A API GraphQL expõe um único endpoint POST /graphql, com queries equivalentes para A, B e C, retornando os mesmos dados (tipos Game, User, Review).

Ambas acessam a mesma base de dados em memória, com lógica de consulta equivalente, para não enviesar a comparação entre REST e GraphQL.

2.3 Scripts de medição

Foi definido um script de medição (`measure_requests.py`) que recebe como parâmetros:

- tipo de API (rest ou graphql),
- cenário (A, B ou C),
- número de repetições (padrão: 30).

Para cada combinação, o script envia as requisições correspondentes, mede o tempo de resposta em milissegundos, calcula o tamanho do corpo em bytes e grava os resultados em um CSV com as colunas: `api_type`, `scenario`, `repetition`, `response_time_ms`, `response_size_bytes`.

2.4 Dados de teste

Os dados de teste foram gerados de forma sintética em memória, simulando um catálogo realista: cerca de **5.000 jogos, 1.000 usuários e 20.000 avaliações**.

Tanto a API REST quanto a GraphQL consomem exatamente essa mesma base de dados simulada.