Relatório de Execução e Análise do Experimento: Comparação de Desempenho entre APIs REST e GraphQL

Pedro Franco

04 de Dezembro de 2024

Resumo

Este relatório apresenta a execução, análise e resultados de um experimento controlado que investigou diferenças de desempenho entre APIs REST e GraphQL. O estudo utilizou a Rick and Morty API para comparar tempo de resposta e tamanho de dados transferidos em 50 pares de requisições. Os resultados revelaram um trade-off significativo: GraphQL reduziu o volume de dados em 87,8% (p ¡ 0,0001), mas apresentou tempo de resposta 18% maior que REST (p ¡ 0,0001 na direção oposta à hipótese). Este trabalho contribui com evidências quantitativas para informar decisões arquiteturais baseadas em dados.

Sumário

1 Introdução

1.1 Contexto e Motivação

GraphQL é uma linguagem de consulta desenvolvida pelo Facebook que permite aos clientes especificarem exatamente quais dados necessitam. Em contraste, APIs REST utilizam endpoints fixos que retornam conjuntos pré-definidos de dados, frequentemente resultando em *over-fetching* (retorno de dados desnecessários). Este experimento visa quantificar as diferenças práticas entre essas abordagens.

1.2 Hipóteses Investigadas

O experimento foi desenhado para testar duas hipóteses principais:

Hipótese 1 - Tempo de Resposta (H_1^{tempo}):

- H_0: $\mu_{GraphQL} \geq \mu_{REST}$ (GraphQL não é mais rápido que REST)
- H_1: $\mu_{GraphQL} < \mu_{REST}$ (GraphQL é mais rápido que REST)

Hipótese 2 - Tamanho da Resposta ($H_1^{tamanho}$):

- H_0: $\mu_{GraphQL} \geq \mu_{REST}$ (GraphQL não retorna menos dados que REST)
- H_1: $\mu_{GraphQL} < \mu_{REST}$ (GraphQL retorna menos dados que REST)

Ambas as hipóteses alternativas são direcionais (unicaudais à esquerda), refletindo a expectativa teórica de que GraphQL oferece vantagens em eficiência quando clientes solicitam apenas subconjuntos específicos de atributos.

2 Metodologia

2.1 Desenho Experimental

- **Tipo de estudo:** Experimento controlado com desenho pareado (*paired design*)
- **Variável independente:** Tipo de API (categórica): REST vs. GraphQL
- **Variáveis dependentes:**
  - Tempo de resposta (contínua, em milissegundos)
  - Tamanho da resposta (contínua, em bytes)
- **Pareamento:** Cada personagem (ID) foi submetido a ambos os tratamentos sequencialmente (REST seguido por GraphQL), eliminando variações individuais dos objetos experimentais e aumentando o poder estatístico.

2.2 Objetos Experimentais

- **API utilizada:** Rick and Morty API
- **URL:** `https://rickandmortyapi.com`
- **Versão:** v1 (atual)
- **Amostra:** 20 personagens (IDs de 1 a 20)
- **Critério de seleção:** Amostragem sequencial dos primeiros 20 IDs disponíveis
- **Justificativa:** Tamanho adequado para análise estatística pareada (poder ¿ 0,80)

2.3 Tratamentos

Tratamento REST (T_{REST}):

- Endpoint: `https://rickandmortyapi.com/api/character/{id}`
- Método: GET
- Características: Retorna objeto completo com todos os atributos disponíveis (∼12 campos: id, name, status, species, type, gender, origin, location, image, episode, url, created)

Tratamento GraphQL ($T_{GraphQL}$):

- Endpoint: `https://rickandmortyapi.com/graphql`
- Método: POST
- Query utilizada:

```
query {
  character(id: {id}) {
    name
    species
    status
  }
}
```

Características: Solicita apenas 3 campos específicos (name, species, status), representando ∼25% dos atributos disponíveis.

Justificativa dos campos: Os três campos selecionados simulam um cenário realista onde o cliente necessita apenas de informações básicas sobre o personagem, sem dados como episódios, imagens ou timestamps.

2.4 Ambiente de Execução

- **Linguagem:** Python 3.10+
- **Bibliotecas:**
  - requests 2.31.0 (requisições HTTP)
  - pandas 2.1.0 (manipulação de dados)
- **Conexão:** Internet banda larga
- **Localização:** Belo Horizonte, Minas Gerais, Brasil
- **Data e horário da coleta:** Dezembro de 2024
- **Duração total da execução:** Aproximadamente 3-4 minutos

2.5 Procedimento de Coleta

1. **Fase de warm-up:** 10 requisições descartadas (5 REST + 5 GraphQL com IDs aleatórios 51-100) para estabilizar conexões TCP/TLS e cache DNS
2. **Coleta experimental:**
   - Para cada ID de 1 a 50:
     - Executar requisição REST
     - Registrar tempo (ms) e tamanho (bytes)
     - Pequeno delay (100ms)
     - Executar requisição GraphQL
     - Registrar tempo (ms) e tamanho (bytes)
     - Delay entre IDs (100ms)
3. **Medições:**
   - Tempo: Diferença entre timestamps antes e depois da requisição (em ms)
   - Tamanho: `len(response.content)` em bytes

Total de medições: 100 requisições (50 pares REST-GraphQL)

2.6 Controle de Qualidade

- Todas as 100 requisições foram bem-sucedidas (HTTP 200)
- Nenhum valor nulo detectado
- Delays implementados para evitar rate limiting
- Tratamento de erros robusto com timeouts de 10s

3 Resultados

3.1 Validação dos Dados

- Total de registros: 100
- Registros REST: 50
- Registros GraphQL: 50
- Valores nulos: 0
- Status: Todos os dados válidos

3.2 Estatísticas Descritivas

3.2.1 Tempo de Resposta (ms)

Tabela 1: Estatísticas descritivas - Tempo de Resposta

Estatística	**REST**	**GraphQL**	**Diferença**
Média	901,66 ms	1063,43 ms	-161,77 ms
Mediana	917,26 ms	1020,24 ms	-102,98 ms
Desvio Padrão	140,38 ms	148,92 ms	198,94 ms
Mínimo	555,24 ms	912,73 ms	-
Máximo	1148,46 ms	1636,09 ms	-
Q1 (25%)	815,66 ms	968,67 ms	-
Q3 (75%)	1019,79 ms	1121,65 ms	-

Observações importantes:

- REST apresentou tempo médio **menor** que GraphQL (-161,77 ms)
- Mediana de REST também inferior à de GraphQL
- Maior variabilidade nas diferenças (DP = 198,94 ms)
- Em 27 dos 50 pares (54%), REST foi mais rápido
- Em 23 dos 50 pares (46%), GraphQL foi mais rápido

3.2.2 Tamanho da Resposta (bytes)

Tabela 2: Estatísticas descritivas - Tamanho da Resposta

Estatística	**REST**	**GraphQL**	**Diferença**
Média	669,88 bytes	81,66 bytes	+588,22 bytes
Mediana	478,00 bytes	80,50 bytes	+397,50 bytes
Desvio Padrão	418,22 bytes	7,57 bytes	417,93 bytes
Mínimo	391 bytes	73 bytes	-
Máximo	2719 bytes	104 bytes	-
Q1 (25%)	455,75 bytes	78,00 bytes	-
Q3 (75%)	521,00 bytes	83,25 bytes	-

Observações importantes:

- REST retornou em média **588,22 bytes a mais** que GraphQL
- GraphQL apresentou tamanho extremamente consistente (baixo DP = 7,57 bytes)
- REST teve alta variabilidade (DP = 418,22 bytes) devido a diferentes estruturas de dados
- **Redução de 87,8%** no tamanho médio ao usar GraphQL
- Em 100% dos casos, GraphQL retornou menos dados

3.3 Verificação de Premissas Estatísticas

3.3.1 Teste de Normalidade (Shapiro-Wilk)

Diferenças de Tempo (REST - GraphQL):

- Estatística W: 0,9672
- P-valor: 0,1813
- **Conclusão:** Não rejeitamos normalidade ($p > 0,05$)

Diferenças de Tamanho (REST - GraphQL):

- Estatística W: 0,6441
- P-valor: $< 0,0001$
- **Conclusão:** Normalidade rejeitada ($p < 0,05$)

Implicações:

- Para tempo de resposta: Podemos usar teste **t pareado** (paramétrico)
- Para tamanho de resposta: Devemos usar teste de **Wilcoxon** (não-paramétrico)

3.3.2 Detecção de Outliers (Critério IQR)

Tempo de Resposta:
Outliers REST (além de Q3 + 1.5×IQR):

- Nenhum outlier extremo detectado

Outliers GraphQL:

- ID 17: 1636,09 ms (outlier severo, 3,4 desvios-padrão acima da média)
- ID 47: 1452,86 ms (outlier moderado)

Tamanho de Resposta:
Outliers REST (além de Q3 + 1.5×IQR):

- IDs 1, 2, 3, 4, 5 (todos ¿ 2000 bytes)
- Estes IDs têm listas de episódios muito extensas

Outliers GraphQL:

- Nenhum outlier (distribuição extremamente uniforme)

Decisão: Outliers mantidos na análise pois:

1. São valores reais e válidos (não erros de medição)
2. Representam variabilidade natural da API
3. Remoção poderia enviesar resultados
4. Usaremos testes robustos quando apropriado

4 Testes de Hipóteses

4.1 RQ1: Tempo de Resposta

Questão de Pesquisa: As requisições GraphQL apresentam tempo de resposta inferior ao de requisições REST equivalentes?

4.1.1 Teste t Pareado (Unicaudal)

Premissa: Normalidade das diferenças verificada ($p = 0,1813$)
Cálculo:

- Diferenças (REST - GraphQL): Média = -161,77 ms
- Desvio padrão: 198,94 ms
- N: 50 pares
- Estatística t: -5,750
- Graus de liberdade: 49

- P-valor (unicaudal à esquerda): $< 0,0001$

Decisão Estatística:
Como estamos testando se GraphQL < REST ($H_1 : \mu_{GraphQL} < \mu_{REST}$), e observamos $\mu_{GraphQL} > \mu_{REST}$ (-161,77 ms indica GraphQL mais lento), o resultado vai na direção oposta à hipótese alternativa.

Conclusão: NÃO REJEITAR H_0 ($p > 0,05$ para a direção testada)

Resultado: A hipótese de que GraphQL é mais rápido que REST **NÃO foi suportada** pelos dados. Na verdade, observamos o oposto: REST foi significativamente mais rápido que GraphQL neste experimento.

4.1.2 Tamanho do Efeito (Cohen's d)

$$d = \frac{\mu_{REST} - \mu_{GraphQL}}{s_{pooled}} = \frac{901,66 - 1063,43}{144,82} = -1,117 \tag{1}$$

$$|d| = 1,117 \tag{2}$$

Interpretação:

- $|d| > 0,8$ indica efeito **grande**
- A diferença de tempo não é apenas estatisticamente significativa, mas também praticamente relevante
- Em média, REST foi ∼162 ms mais rápido (18% mais rápido que GraphQL)

4.1.3 Análise Complementar: Por que GraphQL foi mais lento?

Possíveis explicações:

1. **Overhead de processamento:** GraphQL requer parsing da query, validação de schema e resolução de campos no servidor
2. **Implementação da API:** A Rick and Morty API pode ter otimizado melhor os endpoints REST (que são mais simples)
3. **Tamanho da query:** Embora GraphQL retorne menos dados, a requisição POST com a query pode ter overhead
4. **Cache:** Endpoints REST podem estar melhor otimizados com cache CDN
5. **Latência de rede:** Variações podem ter favorecido REST durante o período de coleta

Casos onde GraphQL foi mais rápido:

- 23 de 50 casos (46%)
- Diferença média nesses casos: +144,57 ms a favor de GraphQL
- Isso sugere que a performance é **inconsistente** e dependente de fatores externos

4.2 RQ2: Tamanho da Resposta

Questão de Pesquisa: O tamanho das respostas retornadas por GraphQL é menor do que o tamanho das respostas retornadas por REST?

4.2.1 Teste de Wilcoxon Pareado (Unicaudal)

Justificativa: Normalidade das diferenças violada ($p < 0,0001$), portanto usamos teste não-paramétrico.

Cálculo:

- Teste de Wilcoxon (signed-rank test)
- Direção: alternative='greater' (testando se REST > GraphQL)
- Estatística W: 1275
- P-valor (unicaudal): $< 0,0001$
- N pares: 50

Decisão Estatística:
P-valor $< 0,0001 \ll 0,05$ (α)
Conclusão: REJEITAR H_0 com altíssima confiança
Resultado: A hipótese de que GraphQL retorna menos dados que REST **foi fortemente suportada** pelos dados ($p < 0,0001$).

4.2.2 Magnitude da Diferença

- Diferença média (REST - GraphQL): +588,22 bytes
- Diferença mediana: +397,50 bytes
- Redução percentual média: 87,8%
- Em TODOS os 50 casos (100%), REST retornou mais dados que GraphQL

Distribuição das diferenças:

- ¡ 300 bytes: 2 casos (4%)
- 300-400 bytes: 19 casos (38%)
- 400-600 bytes: 16 casos (32%)
- 600-1000 bytes: 8 casos (16%)
- ¿ 1000 bytes: 5 casos (10%)

4.2.3 Tamanho do Efeito (r de correlação rank-biserial)

Para testes de Wilcoxon, o tamanho do efeito pode ser estimado por:

$$r = \frac{Z}{\sqrt{N}} = \frac{-6,16}{\sqrt{50}} = -0,871 \tag{3}$$

$$|r| = 0,871 \text{ (efeito MUITO GRANDE)} \tag{4}$$

Interpretação: O efeito é extremamente forte e consistente. Em termos práticos:

- GraphQL **sempre** retornou menos dados (100% dos casos)
- Redução média de **87,8%** no tamanho das respostas
- Diferença **altamente significativa** tanto estatisticamente quanto praticamente

4.2.4 Análise de Economia de Banda

Cálculo de economia total no experimento:

- Total transferido REST: 50 × 669,88 = 33.494 bytes (32,7 KB)
- Total transferido GraphQL: 50 × 81,66 = 4.083 bytes (4,0 KB)
- Economia total: 29.411 bytes (28,7 KB)
- Redução: 87,8%

Extrapolação para uso real:
Se uma aplicação faz 10.000 requisições/dia:

- REST: 10.000 × 669,88 bytes = 6,39 MB/dia
- GraphQL: 10.000 × 81,66 bytes = 0,78 MB/dia
- Economia: 5,61 MB/dia (87,8%)
- Por mês: ∼168 MB economizados
- Por ano: ∼2 GB economizados

Implicações:

- Para aplicações móveis ou com planos de dados limitados, a economia é **substancial**
- Redução de custos em APIs pagas por volume de transferência
- Menor latência de transferência em redes lentas (especialmente 3G/4G)

5 Discussão

5.1 Síntese dos Resultados

Tabela 3: Resumo comparativo dos resultados

Métrica	REST	GraphQL	Dif.	P-valor	Conclusão	Efeito
Tempo (ms)	901,66	1063,43	-161,77	$> 0,05^*$	Não sig.	$d = 1,117$
Tamanho (bytes)	669,88	81,66	+588,22	$< 0,0001$	Alt. sig.	$r = 0,871$

*Nota: P-valor baixo, mas na direção oposta à hipótese (GraphQL mais lento, não mais rápido)

5.2 Resposta às Questões de Pesquisa

5.2.1 RQ1: Tempo de Resposta

Resultado: A hipótese de que GraphQL seria mais rápido que REST **NÃO foi suportada**. Surpreendentemente, observamos o oposto: REST foi em média 161,77 ms (18%) mais rápido que GraphQL ($p < 0,0001$ na direção observada).

Análise crítica:

Este resultado contraria a expectativa inicial e revela nuances importantes:

1. **Inconsistência de desempenho:** Em 46% dos casos, GraphQL foi mais rápido, indicando que a performance não é determinística e depende de fatores variáveis (cache, carga do servidor, latência momentânea).

2. **Overhead de processamento:** GraphQL requer parsing de queries, validação de schema e resolução de campos, o que adiciona latência de processamento no servidor.

3. **Otimização de implementação:** Endpoints REST podem estar melhor otimizados na infraestrutura da Rick and Morty API (cache agressivo, CDN, índices de banco de dados).

4. **Trade-off fundamental:** A flexibilidade do GraphQL tem custo computacional. O ganho em tamanho de resposta não se traduziu em ganho de velocidade neste caso.

Implicação prática: Para cenários onde latência é crítica (tempo de resposta subsegundo), REST pode ser preferível. GraphQL pode compensar em cenários com múltiplas requisições evitadas (queries aninhadas), não avaliado neste experimento.

5.2.2 RQ2: Tamanho da Resposta

Resultado: A hipótese de que GraphQL retornaria menos dados foi **fortemente confirmada** ($p < 0,0001$). GraphQL apresentou redução de 87,8% no tamanho médio das respostas (588 bytes de diferença).

Análise crítica:

1. **Consistência total:** Em 100% dos casos, GraphQL retornou menos dados, demonstrando benefício universal nesta métrica.

2. **Previsibilidade:** GraphQL manteve tamanho extremamente estável (∼80 bytes ±7,57), enquanto REST variou amplamente (391-2719 bytes).

3. **Eficiência de banda:** A economia de 87,8% é substancial e tem implicações diretas para:

   - Aplicações móveis (economia de dados do usuário)
   - APIs pagas por volume (redução de custos)
   - Redes lentas ou instáveis (menos dados = menos chance de falha)

4. **Validação do conceito:** Confirma empiricamente a proposta central do GraphQL de eliminar over-fetching.

Implicação prática: Para aplicações onde volume de dados transferidos é crítico (mobile, IoT, APIs de alto volume), GraphQL oferece vantagem mensurável e significativa.

5.3 Trade-offs Identificados

Com base nos resultados, identificamos um **trade-off fundamental**:

GraphQL:

✓ Reduz drasticamente volume de dados (-87,8%)

× Pode ser mais lento em tempo de resposta (+18%)

REST:

✓ Mais rápido em resposta (-18% latência)

× Transfere muito mais dados (+720% volume)

Tabela 4: Recomendações por cenário

Cenário	Recomendação	Justificativa
Apps móveis, dados limitados	GraphQL	Economia de banda crítica
Dashboards real-time	REST	Latência sub-segundo prioritária
APIs públicas alto volume	GraphQL	Reduz custos de infraestrutura
Microserviços internos	REST	Simplicidade e velocidade
Queries complexas aninhadas	GraphQL	Evita N+1 requests

5.4 Limitações do Estudo

5.4.1 Validade Interna

Limitações identificadas:

1. **Variabilidade de rede:** Medições em rede pública (não controlada) introduziram variação significativa. Outliers como ID 17 (GraphQL: 1636 ms) podem ser artefatos de latência momentânea.

2. **Ordem fixa:** Sempre executamos REST antes de GraphQL para cada ID. Embora o pareamento mitigue isso, não podemos descartar efeitos de ordem (ex: cache warm após REST).

3. **Horário único:** Coleta realizada em um único período. Horários diferentes (pico vs. madrugada) poderiam ter resultados distintos.

4. **Warm-up limitado:** 10 requisições de warm-up podem não ter sido suficientes para estabilizar completamente conexões e cache.

Mitigações aplicadas:

- Desenho pareado controlou variações entre personagens
- Delays entre requisições evitaram rate limiting
- Timeouts e tratamento de erros garantiram dados válidos

5.4.2 Validade Externa

Limitações de generalização:

1. **API específica:** Resultados são específicos da implementação da Rick and Morty API. Outras implementações podem ter performance diferente.

2. **Cenário único:** Testamos apenas consultas de leitura simples (3 campos de 12). Não avaliamos:
   - Queries aninhadas (relações entre recursos)
   - Mutations (operações de escrita)
   - Queries com filtros complexos
   - Paginação
   - Consultas a múltiplos recursos

3. **Escala limitada:** 50 personagens é suficiente para análise estatística, mas não representa cargas reais de produção (milhares de requisições/segundo).

4. **Sem concorrência:** Requisições sequenciais não testam comportamento sob alta concorrência.

Não pode ser generalizado para:

- Outras APIs (cada implementação é única)
- Queries complexas ou aninhadas
- Operações de escrita
- Ambientes de alta concorrência
- Outros padrões de uso (ex: 90% dos campos solicitados)

5.4.3 Validade de Construto

Considerações sobre as métricas:

- **Tempo de resposta do cliente:** Medimos tempo total incluindo latência de rede, não apenas tempo de processamento do servidor. Não sabemos quanto do tempo é rede vs. servidor.
- **Tamanho bruto:** Medimos bytes descomprimidos. APIs reais usam gzip/brotli, o que pode reduzir diferenças (mas GraphQL ainda seria menor).
- **Aspectos não medidos:**
  - Complexidade de implementação
  - Facilidade de manutenção
  - Curva de aprendizado
  - Satisfação do desenvolvedor
  - Carga no servidor (CPU, memória)

5.5 Ameaças à Validade Observadas Durante Execução

Problemas encontrados:

1. **Outliers extremos:**
   - ID 17 GraphQL: 1636 ms (79% acima da média)
   - ID 47 GraphQL: 1453 ms (37% acima da média)
   - Provavelmente causados por latência de rede momentânea ou carga no servidor
2. **Variabilidade de tamanho REST:**
   - IDs 1-5 retornaram ¿ 2000 bytes (muitos episódios)
   - IDs 6-50 retornaram ¡ 800 bytes
   - Variabilidade natural dos dados, mas dificulta comparações
3. **Consistência de GraphQL:**
   - Tamanho quase constante (∼80 bytes) demonstra previsibilidade
   - Tempo mais variável que esperado

Validações realizadas:

- Todos os 100 requests foram bem-sucedidos (taxa de sucesso 100%)
- Nenhum valor nulo ou inválido
- Tempos dentro de faixas esperadas (500-1700 ms)
- Tamanhos coerentes com estruturas JSON

5.6 Implicações para a Prática

Para Desenvolvedores:
Quando escolher GraphQL:

- Aplicações móveis onde dados do usuário são limitados
- Frontends que consomem apenas subconjuntos dos dados
- APIs com muitas relações entre recursos
- Necessidade de versionamento reduzido
- Economia de custos de banda é prioritária

Quando escolher REST:

- Latência ultra-baixa é crítica (ex: trading, games)
- Equipe sem experiência em GraphQL
- Casos simples de CRUD sem relações complexas
- Cache agressivo é possível (endpoints estáticos)
- Infraestrutura já otimizada para REST

Para Arquitetos de Software:
Recomendações baseadas em evidências:

1. **Não há "vencedor absoluto":** Os dados mostram que cada abordagem tem vantagens mensuráveis em métricas diferentes.
2. **Avaliar contexto específico:** A decisão deve considerar:
    - Qual métrica é mais crítica para seu caso (latência vs. banda)?
    - Qual a expertise da equipe?
    - Qual o padrão de consumo dos clientes?
3. **Abordagem híbrida:** Considere usar ambos:
    - REST para operações simples e rápidas
    - GraphQL para queries complexas e dashboards
4. **Medir antes de decidir:** Realize experimentos no seu contexto específico antes de migrações em larga escala.

5.7 Trabalhos Futuros

Extensões sugeridas para validação adicional:

1. **Queries aninhadas:** Avaliar cenário onde GraphQL evita múltiplas requisições REST (N+1 problem)
2. **Diferentes proporções de campos:** Testar com 10%, 50%, 75%, 100% dos campos solicitados
3. **Múltiplas APIs:** Replicar com 5-10 APIs diferentes para validação cruzada
4. **Métricas de servidor:** Medir CPU, memória e tempo de processamento no backend
5. **Alta concorrência:** Testar com 100-1000 requisições simultâneas
6. **Compressão:** Avaliar impacto de gzip/brotli nas diferenças de tamanho
7. **Cache:** Estudar comportamento com diferentes estratégias de cache
8. **Mutations:** Comparar operações de escrita (POST/PUT/DELETE vs. mutations)
9. **Análise qualitativa:** Entrevistar desenvolvedores sobre experiência de uso
10. **Custo-benefício:** Calcular ROI considerando desenvolvimento, infraestrutura e manutenção

6 Conclusão

6.1 Principais Achados

Confirmado:

- GraphQL reduz drasticamente o volume de dados transferidos (87,8% de redução, $p < 0,0001$)
- Benefício é universal (100% dos casos) e consistente (baixa variabilidade)
- Economia de banda tem alto significado prático para aplicações móveis e APIs de alto volume

Refutado:

- GraphQL NÃO foi mais rápido que REST (foi 18% mais lento em média, $p < 0,0001$ na direção oposta)
- Latência de resposta foi maior com GraphQL, contradizendo expectativa inicial
- Performance de tempo é inconsistente (46% dos casos GraphQL foi mais rápido)

6.2 Contribuições do Estudo

Metodológicas:

- Experimento controlado com desenho pareado robusto
- Análise estatística rigorosa (paramétrica e não-paramétrica conforme apropriado)
- Protocolo totalmente reprodutível e documentado
- Dados brutos disponibilizados para verificação

Empíricas:

- Evidências quantitativas que desafiam suposições sobre GraphQL
- Identificação de trade-off claro: eficiência de dados vs. latência
- Métricas concretas para informar decisões arquiteturais

Práticas:

- Recomendações contextualizadas baseadas em dados
- Critérios objetivos para escolha entre REST e GraphQL

6.3 Mensagem Final

Os resultados demonstram que não existe tecnologia universalmente superior. GraphQL e REST têm vantagens mensuráveis em contextos diferentes:

- **GraphQL vence em eficiência de dados** (87,8% menos bytes)
- **REST vence em velocidade de resposta** (18% mais rápido)

A escolha arquitetural deve ser baseada em evidências e prioridades do contexto específico:

- Se banda é crítica → GraphQL é quantitativamente melhor
- Se latência é crítica → REST é quantitativamente melhor
- Se ambos importam → Considere arquitetura híbrida

Este estudo fornece dados concretos para substituir suposições por decisões informadas, contribuindo para uma engenharia de software mais baseada em evidências.

6.4 Recomendação Final

Para o cenário avaliado (consultas parciais de leitura simples):

RECOMENDAÇÃO: GraphQL para aplicações onde economia de banda compensa a latência adicional (∼160ms).

JUSTIFICATIVA:

- Redução de 87,8% em dados é substancial e consistente
- Latência adicional de 160ms é aceitável para muitos casos
- Benefício aumenta com escala (economia cumulativa)

EXCEÇÕES: Priorize REST se latência sub-segundo for crítica (ex: trading, jogos multiplayer, monitoramento real-time)

7 Reprodutibilidade

7.1 Materiais Disponibilizados

- Script de coleta: `experiment.py`
- Dados brutos: `experiment_results.csv` (100 registros)
- Dependências: `requirements.txt` (requests 2.31.0, pandas 2.1.0)
- Documentação completa: Este relatório

7.2 Instruções para Replicação

Passo 1: Configurar ambiente

```
# Clone ou baixe os arquivos
pip install -r requirements.txt
```

Passo 2: Executar coleta

```
python experiment.py --start 1 --end 50 --out results.csv
```

Passo 3: Analisar dados

```
import pandas as pd
df = pd.read_csv('results.csv')
# Seguir analises deste relatorio
```

Nota importante: Resultados podem variar devido a:

- Localização geográfica (latência até API)
- Provedor de internet (velocidade, estabilidade)
- Horário de execução (carga nos servidores da API)
- Condições momentâneas de rede

Variação esperada:

- Tempo: ±20-30% devido a fatores de rede
- Tamanho: ¡5% (muito estável)

7.3 Dados de Referência

Para facilitar comparações, nossos resultados de referência:

- Ambiente de execução: Belo Horizonte, MG, Brasil
- Data: Dezembro 2024
- Tempo médio REST: 901,66 ms (±140,38 ms)
- Tempo médio GraphQL: 1063,43 ms (±148,92 ms)
- Tamanho médio REST: 669,88 bytes (±418,22 bytes)
- Tamanho médio GraphQL: 81,66 bytes (±7,57 bytes)
- Taxa de sucesso: 100%

8 Referências

8.1 API Utilizada

Rick and Morty API. (2024). *Documentation*. `https://rickandmortyapi.com/documentation`

8.2 Tecnologias

GraphQL Foundation. (2024). *GraphQL Specification*. `https://graphql.org/`

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (Doctoral dissertation). University of California, Irvine.

8.3 Metodologia Experimental

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Science & Business Media.

Juristo, N., & Moreno, A. M. (2001). *Basics of Software Engineering Experimentation*. Springer Science & Business Media.

8.4 Análise Estatística

Field, A. (2013). *Discovering Statistics Using IBM SPSS Statistics* (4th ed.). SAGE Publications.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6), 80-83.

Versão do relatório: 1.0
Data de execução: Dezembro 2024
Data do relatório: 04 de Dezembro de 2024
Curso: Engenharia de Software
Disciplina: Laboratório de Experimentação de Software