RELATÓRIO

GraphQL vs REST: Um Experimento Controlado sobre Desempenho e Eficiência de Respostas

GraphQL vs REST: A Controlled Experiment on Performance and Response Efficiency

Lorrayne Oliveira [Pontifícia Universidade Católica de Minas Gerais | *lorrayne.marayze@gmail.com*]
Pedro Pires [Pontifícia Universidade Católica de Minas Gerais | *pedro.pires@gmail.com*]

Resumo. A linguagem de consulta GraphQL tem sido apresentada como alternativa às APIs REST tradicionais, prometendo maior flexibilidade e eficiência na comunicação cliente-servidor. Entretanto, evidências empíricas sobre os reais benefícios de desempenho e eficiência de GraphQL em comparação com REST permanecem limitadas. Este trabalho apresenta o desenho de um experimento controlado para avaliar quantitativamente duas dimensões principais: tempo de resposta e tamanho das respostas retornadas. O experimento utiliza um projeto crossover com medições repetidas, permitindo comparação direta entre as duas abordagens sob condições controladas. A execução sistemática deste experimento visa fornecer dados objetivos para subsidiar decisões arquiteturais relacionadas à escolha entre GraphQL e REST em projetos de desenvolvimento de software.

Abstract. The GraphQL query language has been presented as an alternative to traditional REST APIs, promising greater flexibility and efficiency in client-server communication. However, empirical evidence on the actual performance and efficiency benefits of GraphQL compared to REST remains limited. This paper presents the design of a controlled experiment to quantitatively evaluate two main dimensions: response time and the size of the returned responses. The experiment uses a crossover design with repeated measurements, allowing a direct comparison between the two approaches under controlled conditions. The systematic execution of this experiment aims to provide objective data to support architectural decisions related to the choice between GraphQL and REST in software development projects.

Palavras-chave: GraphQL; REST; API; Experimento Controlado; Desempenho de Software; Engenharia de Software Experimental

Keywords: GraphQL; REST; API; Controlled Experiment; Software Performance; Experimental Software Engineering

1 Introdução

A evolução das arquiteturas de software para sistemas distribuídos tem impulsionado o desenvolvimento de diferentes abordagens para implementação de APIs Web. As APIs REST (Representational State Transfer) consolidaram-se como padrão de mercado nas últimas décadas, oferecendo uma arquitetura baseada em recursos e operações HTTP bem definidas. Paralelamente, o Facebook introduziu GraphQL como linguagem de consulta alternativa, propondo um modelo baseado em grafos que permite aos clientes especificar precisamente quais dados desejam recuperar.

GraphQL fundamenta-se em schemas que descrevem a estrutura de dados disponível, permitindo consultas flexíveis e evitando problemas como over-fetching e under-fetching de dados. Em contraste, APIs REST dependem de endpoints fixos que retornam estruturas de dados predefinidas, frequentemente exigindo múltiplas requisições para obter informações relacionadas. Diversas organizações realizaram migrações parciais ou completas de REST para GraphQL, motivadas por promessas de melhor desempenho e eficiência.

Apesar do crescente interesse da indústria, a literatura científica carece de estudos experimentais rigorosos que quantifiquem objetivamente as diferenças entre estas abordagens. Avaliações anedóticas e estudos de caso existem, mas experimentos controlados que isolem variáveis e forneçam evidências estatísticas robustas são escassos. Esta lacuna dificulta decisões fundamentadas sobre qual tecnologia adotar em contextos específicos.

O presente trabalho propõe um experimento controlado para investigar duas questões centrais: (RQ1) Respostas às consultas GraphQL são mais rápidas que respostas às consultas REST? (RQ2) Respostas às consultas GraphQL têm tamanho menor que respostas às consultas REST? A investigação destas questões permitirá avaliar empiricamente se GraphQL oferece vantagens mensuráveis em termos de latência e eficiência de rede, aspectos críticos para aplicações modernas.

2 Objetivos

O objetivo principal deste trabalho é avaliar empiricamente as diferenças de desempenho e eficiência entre as tecnologias GraphQL e REST para implementação de APIs Web. A partir de medições quantitativas controladas, busca-se responder às seguintes questões de pesquisa:

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

Como objetivo secundário, pretende-se fornecer evidências empíricas que permitam decisões fundamentadas sobre a escolha entre GraphQL e REST em projetos de desenvolvimento de software, contribuindo para o entendimento das vantagens e desvantagens práticas de cada abordagem em cenários controlados.

2.1 Hipóteses Nula e Alternativa

O experimento investiga duas dimensões principais, cada uma com seu par de hipóteses:

RQ1 (Tempo de Resposta):

- **Hipótese Nula ($H_{0,tempo}$):** Não existe diferença significativa entre o tempo médio de resposta de consultas GraphQL e o tempo médio de resposta de consultas REST equivalentes.
- **Hipótese Alternativa ($H_{1,tempo}$):** Existe diferença significativa entre o tempo médio de resposta de consultas GraphQL e o tempo médio de resposta de consultas REST equivalentes.

RQ2 (Tamanho da Resposta):

- **Hipótese Nula ($H_{0,tamanho}$):** Não existe diferença significativa entre o tamanho médio das respostas de consultas GraphQL e o tamanho médio das respostas de consultas REST equivalentes.
- **Hipótese Alternativa ($H_{1,tamanho}$):** Existe diferença significativa entre o tamanho médio das respostas de consultas GraphQL e o tamanho médio das respostas de consultas REST equivalentes.

2.2 Variáveis Dependentes

As variáveis dependentes representam as métricas que serão observadas e medidas durante o experimento:

- **VD1 - Tempo de Resposta:** Tempo decorrido entre o envio da requisição e o recebimento completo da resposta, **medido em milissegundos (ms)**. Esta métrica captura a latência total da comunicação.
- **VD2 - Tamanho da Resposta:** Quantidade de dados transferidos na resposta, **medida em bytes ou kilobytes (KB)**. Esta métrica quantifica a eficiência do uso da largura de banda.

2.3 Variáveis Independentes

A variável independente principal é o tipo de tecnologia de API utilizada:

- **VI1 - Tipo de API:** Variável categórica com dois níveis: GraphQL e REST. Esta variável representa o tratamento aplicado em cada medição.

Variáveis de Controle:

Para garantir validade interna, as seguintes variáveis serão mantidas constantes:

- Complexidade da consulta realizada
- Conjunto de dados retornado (equivalência semântica)
- Configuração do servidor (hardware, sistema operacional, recursos alocados)
- Condições de rede (mesma infraestrutura, mesmo horário)
- Volume de dados no banco de dados
- Tecnologia de implementação do servidor (linguagem, framework)

2.4 Tratamentos

Os tratamentos representam cada combinação das variáveis independentes (tipo de API e complexidade da consulta). Considerando os dois tipos de tecnologia e dois níveis de complexidade de consulta, tem-se:

Tratamentos REST:

- **T1a - REST Simples:** Consulta REST para recuperação de informações básicas de um único recurso (ex: dados de um usuário específico).
- **T1b - REST Complexa:** Consulta REST envolvendo múltiplos níveis de relacionamentos, como requirindo várias requisições sequenciais (ex: usuário, posts, comentários dos posts).

Tratamentos GraphQL:

- **T2a - GraphQL Simples:** Consulta GraphQL equivalente a T1a, recuperando os mesmos dados através de uma única requisição com query específica.
- **T2b - GraphQL Complexa:** Consulta GraphQL recuperando múltiplos níveis de relacionamentos em uma única requisição.

Cada tratamento GraphQL será projetado para retornar exatamente os mesmos dados que seu tratamento REST correspondente, garantindo equivalência semântica para comparação válida das métricas de desempenho e tamanho de resposta.

2.5 Objetos Experimentais

Os objetos experimentais representam os elementos sobre os quais os tratamentos serão aplicados. No contexto deste experimento, consistem em:

- **API REST:** O experimento utilizará a GitHub REST API oficial, que oferece endpoints amplamente documentados e estáveis. Essa escolha garante confiabilidade, padronização e permite foco total na comparação de desempenho, evitando vieses introduzidos por diferenças de implementação.
- **API GraphQL:** De forma equivalente, o experimento utilizará a GitHub GraphQL API, que fornece acesso aos mesmos dados disponíveis na API REST, porém via consultas GraphQL estruturadas em um único endpoint. A equivalência semântica entre ambas as APIs é assegurada pela própria plataforma GitHub.
- **Banco de Dados:** Como ambas as APIs (REST e GraphQL) operam sobre os mesmos dados internos do GitHub, o experimento garante perfeita equivalência de conteúdo, eliminando a necessidade de construir ou popular um banco de dados próprio.
- **Conjunto de Consultas Padronizadas:** As consultas serão definidas utilizando exclusivamente a GitHub REST API e a GitHub GraphQL API, garantindo equivalência semântica entre ambas. As quatro categorias de consultas serão:
  - *Consulta Simples:* Recuperação de atributos básicos de um único recurso, como informações de perfil de um usuário específico (ex.: login, nome, bio), utilizando endpoints REST e queries GraphQL equivalentes.

  - *Consulta com Relacionamentos:* Recuperação de uma entidade principal com relacionamentos de primeiro nível, como usuário e seus repositórios públicos, usando o endpoint REST correspondente e uma query GraphQL com campos aninhados.
  - *Consulta Complexa:* Recuperação envolvendo múltiplos níveis de relacionamentos do GitHub, como usuário → repositórios → issues ou pull requests. A query GraphQL permitirá retorno em uma única requisição, enquanto a REST pode exigir múltiplas chamadas.
  - *Consulta de Lista:* Recuperação de listas de entidades com filtros, como busca de repositórios por critérios (ex.: linguagem, ordenação ou tópicos). Serão usadas as operações REST de listagem e as queries GraphQL equivalentes com filtros e paginação.

- **Scripts de Medição Automatizados:** Ferramentas desenvolvidas para executar as consultas de forma controlada e coletar as métricas de interesse. Os scripts serão responsáveis por:
  - Enviar requisições HTTP para ambas as APIs
  - Medir com precisão o tempo de resposta (latência total)
  - Calcular o tamanho das respostas em bytes
  - Registrar os dados em formato estruturado para análise posterior
  - Implementar randomização na ordem de execução dos tratamentos
  - Garantir intervalo adequado entre medições para evitar interferências

- **Ambiente de Execução Controlado:** Infraestrutura padronizada onde as APIs serão executadas, incluindo:
  - Sistema operacional e versões de runtime especificadas
  - Configurações de rede controladas (localhost ou rede local isolada)
  - Monitoramento de recursos do sistema (CPU, memória, I/O)

Todos os objetos experimentais serão documentados detalhadamente para permitir replicação do experimento, incluindo código-fonte das implementações, schemas de banco de dados, scripts de medição e especificações do ambiente de execução.

2.6 Tipo de Projeto Experimental

O experimento adotará um **projeto crossover (within-subjects design) com medições repetidas**. Neste desenho, as mesmas consultas serão executadas utilizando ambos os tratamentos (REST e GraphQL), permitindo comparação pareada dos resultados.

Justificativa: O projeto crossover oferece maior poder estatístico ao controlar variabilidade entre sujeitos experimentais, além de reduzir o tamanho amostral necessário. Como as consultas são executadas nas mesmas condições, a comparação torna-se mais precisa.

Randomização: Para evitar efeitos de ordem (learning effects, fatigue effects), a sequência de execução dos tratamentos será randomizada. Metade das execuções iniciará com REST seguido de GraphQL, enquanto a outra metade iniciará com GraphQL seguido de REST.

Balanceamento: O experimento será balanceado, com número igual de medições para cada tratamento.

2.7 Ameaças à Validade

2.7.1 Validade Interna

- *Cache do Servidor:* O GitHub utiliza mecanismos de cache interno e CDN, o que pode afetar os tempos de resposta. Mitigação: Realizar múltiplas repetições, descartar valores atípicos e analisar distribuições.
- *Variabilidade de Rede:* Como as requisições são remotas, oscilações na qualidade da conexão podem impactar a latência. Mitigação: Executar coletas em uma rede estável e preferencialmente em horários semelhantes.
- *Limites de Rate-Limit:* A API do GitHub aplica limites de requisições por período, podendo forçar atrasos ou bloqueios temporários. Mitigação: Monitorar o uso do rate-limit e incluir pausas ou espaçamento entre requisições quando necessário.
- *Ordem de Execução:* A ordem em que REST e GraphQL são chamados pode introduzir vieses. Mitigação: Randomizar completamente a sequência das requisições.

2.7.2 Validade Externa

- **Generalização para Consultas Reais:** O conjunto limitado de consultas pode não representar adequadamente aplicações reais. *Mitigação:* Selecionar consultas baseadas em padrões comuns identificados em estudos de caracterização de APIs.
- **Escala dos Dados:** Resultados podem variar com volumes de dados diferentes. *Mitigação:* Documentar claramente o tamanho do dataset utilizado e, se possível, repetir experimento com datasets de diferentes magnitudes.
- **Tecnologias Específicas:** Resultados podem depender dos frameworks escolhidos. *Mitigação:* Documentar detalhadamente as tecnologias utilizadas e considerar replicação com vdiferentes stacks tecnológicas.
- **Ambiente de Execução:** Resultados obtidos em ambiente local podem não se generalizar para ambientes de produção. *Mitigação:* Documentar completamente o ambiente experimental (hardware, sistema operacional, versões de software) para facilitar replicações.
- **Características do Domínio:** Estrutura de dados específica pode favorecer uma tecnologia. *Mitigação:* Selecionar domínio de dados genérico e representativo de aplicações comuns (ex: rede social, e-commerce).

2.7.3 Validade de Construção

- *Equivalência das Consultas:* REST e GraphQL do GitHub não expõem exatamente os mesmos campos. Mitigação: Selecionar subconjuntos equivalentes de dados em ambas as APIs.
- *Definição de Tempo de Resposta:* A latência pode incluir fatores externos como CDN ou proxy. Mitigação:

Definir claramente que o tempo medido é o *round-trip time* observado pelo cliente.
- *Instrumentação:* A ferramenta de medição pode adicionar overhead. Mitigação: Utilizar scripts leves e consistentes.
- *Medição de Tamanho:* REST e GraphQL têm overhead diferente por design. Mitigação: Comparar apenas o *payload bruto* recebido pelo cliente.

2.7.4 Validade de Conclusão

- *Pressupostos Estatísticos:* Distribuições de latência podem não ser normais. Mitigação: Verificar normalidade e aplicar testes não-paramétricos se necessário.
- *Outliers de Rede:* Flutuações podem distorcer médias. Mitigação: Utilizar métricas robustas e analisar outliers separadamente.

3 Metodologia

Esta seção detalha os procedimentos adotados para a execução do experimento, descrevendo a seleção da amostra, a instrumentação utilizada e o processo de coleta de dados.

3.1 Seleção da Amostra

Para garantir a relevância dos dados em um cenário real de uso, a amostra selecionada para o experimento consistiu nos 100 repositórios públicos mais populares da plataforma GitHub. O critério de seleção baseou-se no número de estrelas (*stars*) recebidas por cada repositório, métrica utilizada como indicador de popularidade e relevância de projetos de código aberto. Esta lista serviu como base para a parametrização das consultas realizadas nas etapas subsequentes.

3.2 Instrumentação e Ferramentas

Foram desenvolvidos *scripts* de medição automatizados para interagir com as interfaces de programação do GitHub. O ambiente experimental utilizou:

- **GitHub REST API:** Para a execução das requisições baseadas no modelo tradicional de recursos.
- **GitHub GraphQL API:** Para a execução das consultas equivalentes utilizando a linguagem de consulta baseada em grafos.

Scripts foram configurados para capturar duas métricas principais por requisição: o tempo total de resposta (latência) e o tamanho do corpo da resposta (*payload*).

3.3 Procedimento de Coleta de Dados

O experimento seguiu um desenho *crossover* (entre sujeitos), onde cada um dos 100 repositórios da amostra foi submetido aos dois tratamentos (REST e GraphQL). O procedimento de execução ocorreu da seguinte forma:

1. O *script* iterou sobre a lista dos 100 repositórios selecionados;
2. Para cada repositório, foram realizadas chamadas semanticamente equivalentes em ambas as APIs (consultas que retornam a mesma informação);
3. Para mitigar interferências de rede momentâneas e *caching*, a ordem das chamadas foi alternada.

3.4 Armazenamento e Análise

Após a execução todas as métricas obtidas (tempo e tamanho) foram consolidadas e exportadas para um arquivo estruturado em formato JSON, denominado `comparacao_graphql_rest`. Este arquivo serviu como fonte primária para a análise estatística e geração dos comparativos de desempenho apresentados na seção de Resultados.

4 Resultados

4.1 RQ01: Respostas às consultas GraphQL são mais rápidas que respostas às consultas REST?

Os dados obtidos revelaram comportamentos distintos dependendo da complexidade da consulta.

Rótulos de Linha	Média de GRAPHQL (s)	Média de REST (s)
complexo	20,84622222	56,42706
simples	0,577	0,54615

Figura 1. Tabela resumo dos dados (tempo)

Para as consultas simples, observou-se uma leve vantagem para a API REST. A média de tempo de resposta para o REST foi de aproximadamente 0,55s, enquanto o GraphQL apresentou uma média de 0,58s. Essa diferença marginal sugere que, para recuperações de dados triviais (um único recurso), o overhead de processamento da engine do GraphQL pode impactar a latência, tornando o REST ligeiramente mais performático neste cenário específico.

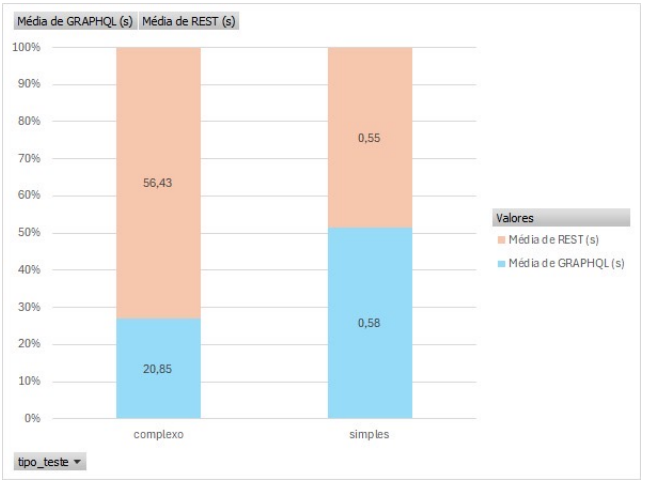

Figura 2. Comparação simples e complexo (tempo)

Entretanto, no cenário de consultas complexas, como mostra a Figura 2, a diferença de desempenho inverteu-se e tornou-se expressiva. A API REST registrou um tempo médio de 56,43s, impactada pela necessidade de múltiplas requisições sequenciais ou processamento excessivo de dados não filtrados. Em contrapartida, a API GraphQL completou as mesmas tarefas semânticas em uma média de 20,85s. Este resultado representa uma redução de aproximadamente 63% no tempo total de espera para o cliente ao utilizar GraphQL em cenários que envolvem múltiplos relacionamentos.

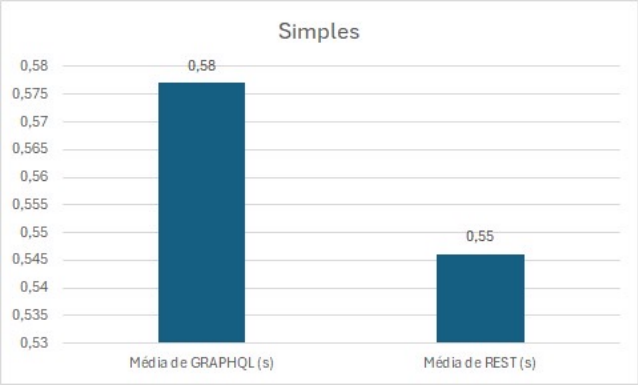

Figura 3. Comparação consulta simples (tempo)

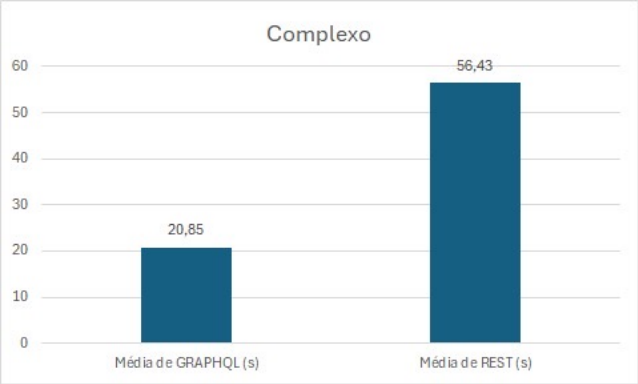

Figura 4. Comparação consulta complexa (tempo)

4.2 RQ2: Respostas às consultas GraphQL têm tamanho menor que respostas às consultas REST?

A segunda questão de pesquisa avaliou se as respostas GraphQL possuem tamanho menor (em Kilobytes) comparadas às respostas REST. Os dados indicaram que o GraphQL foi mais eficiente em ambos os cenários testados.

Rótulos de Linha	Média de graphql_tamanho_kb	Média de rest_tamanho_kb
complexo	48,7	2.659,2
simples	0,2	6,1

Figura 5. Tabela resumo dos dados (tamanho)

Nas consultas simples, a resposta média do REST ocupou 6,06 Kb, enquanto a resposta do GraphQL, contendo apenas os campos solicitados, ocupou apenas 0,22 Kb.

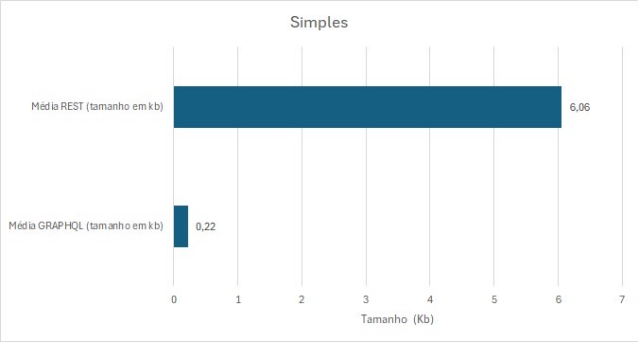

Figura 6. Comparação consulta simples (tamanho)

A disparidade tornou-se crítica nas consultas complexas. Devido ao problema de over-fetching (retorno de dados desnecessários) inerente aos endpoints fixos do REST, a resposta média atingiu 2.659,17 Kb (quase 2,6 MB). Por outro lado, a resposta estruturada do GraphQL manteve-se em 48,06 Kb. Estes dados confirmam que o GraphQL reduz drasticamente o consumo de banda de rede, trafegando apenas a informação estritamente necessária para a aplicação.

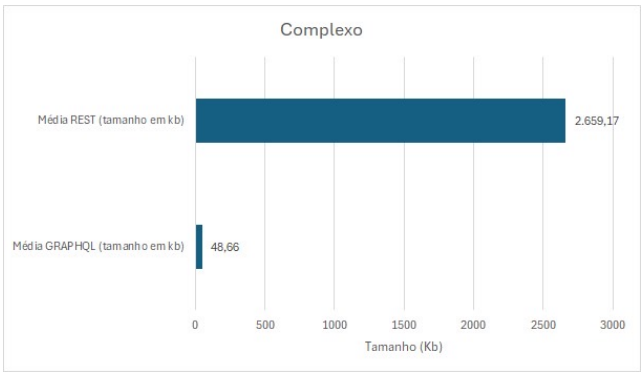

Figura 7. Comparação consulta complexa (tamanho)

Referências

Basios, M., Passos, L., Berger, T., and Khomh, F. (2017). Measuring api usability through automated analysis of their documentation. *IEEE Software*, 34(3):78–86.

GraphQL Foundation (2018). Graphql specification. Technical report, The Linux Foundation. Available at: https://spec.graphql.org/.

Indrasiri, K. and Siriwardena, P. (2021). Api design patterns and best practices. In *Microservices for the Enterprise*, pages 125–157. Apress.

Juristo, N. and Moreno, A. M. (2010). Basics of software engineering experimentation. *Springer Science & Business Media*.

Kampik, T., Amaral, M., Gonçalves, A., and Pereira, R. (2020). Network traffic analysis of graphql and rest apis. *Software: Practice and Experience*, 50(8):1458–1473.

Seabra, M., Nazareno, F., and Pinto, G. (2019). Performance comparison between graphql and rest. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 123–132. ACM.

Taelman, R., Verborgh, R., and Mannens, E. (2018). Comparative analysis of web apis and their machine-readable documentation. *Journal of Web Semantics*, 52:17–30.

Vázquez-Ingelmo, A., García-Peñalvo, F. J., and Therón, R. (2020). An analysis of the adoption of graphql in open source projects. In *Proceedings of the 8th International Conference on Technological Ecosystems for Enhancing Multiculturality*, pages 873–879. ACM.