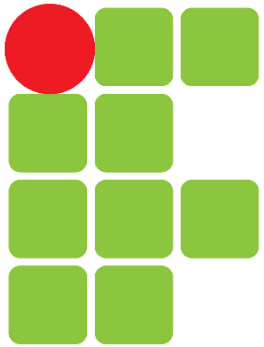
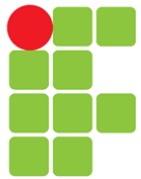


INSTITUTO FEDERAL
SÃO PAULO



Introdução ao Desenvolvimento Backend

Desenvolvimento Web Backend (BRADWBK)

Prof. Luiz Gustavo Diniz de Oliveira Véras

E-mail: gustavo_veras@ifsp.edu.br

Objetivos

- ✓Contextualização
- ✓O que é um Backend?
- ✓Confiabilidade, Escalabilidade, Manutenabilidade
- ✓APIs Web

Contextualização

Se você trabalhou em engenharia de software nos últimos anos, especialmente em sistemas do lado do servidor e de back-end, provavelmente foi bombardeado com uma infinidade de palavras-chave relacionadas ao armazenamento e processamento de dados.

```
NoSQL! Big Data! Web-scale! Sharding! Eventual consistency! ACID!
Cloud services! DevOps! MapReduce! Real-time!
```

Contextualização

Para entende-los, temos que nos aprofundar mais no significado dessas palavras.

Felizmente, por trás das rápidas mudanças na tecnologia, existem princípios duradouros que permanecem verdadeiros, independentemente da versão de uma ferramenta específica que você está usando. Se você entende esses princípios, está em posição de ver onde cada ferramenta se encaixa, como fazer bom uso e como evitar suas armadilhas.

Contextualização

Os principais fatores que impulsionam o desenvolvimento desses novos termos incluem:

- Empresas de tecnologia, como Google, Amazon e Facebook, lidam com grandes volumes de dados e tráfego, exigindo novas ferramentas para gerenciar essa escala de forma eficiente.
- As empresas precisam ser ágeis, testar hipóteses de forma econômica e responder rapidamente às mudanças do mercado, mantendo ciclos de desenvolvimento curtos e modelos de dados flexíveis.
- O software livre e de código aberto tem se tornado a escolha preferida em muitos ambientes, substituindo soluções comerciais ou internas.

Contextualização

Os principais fatores que impulsionam o desenvolvimento desses novos termos incluem:

- As velocidades dos processadores não estão aumentando significativamente, mas os processadores multicore e redes mais rápidas tornam o paralelismo cada vez mais relevante.
- Pequenas equipes podem construir sistemas distribuídos globalmente graças a infraestruturas como Amazon Web Services (IaaS).
- Os serviços precisam ser altamente disponíveis, pois longos períodos de inatividade são cada vez mais inaceitáveis.

Contextualização

- Essas palavras da moda que preenchem esse espaço são um sinal de entusiasmo por novas possibilidades e soluções, o que é uma ótima coisa.
- No entanto, como engenheiros e arquitetos de software, também precisamos ter um entendimento tecnicamente preciso das várias tecnologias e suas compensações se queremos criar boas aplicações.
- Após o curso, espera-se que você esteja um passo adiante na capacidade de decidir que tipo de tecnologia é apropriada para qual finalidade e entender como as ferramentas podem ser combinadas para formar a base de uma boa arquitetura de aplicativos.

O que é um Backend?

Às vezes chamado de lado do servidor, o back-end do aplicativo gerencia a funcionalidade geral do seu aplicativo web. Quando o usuário interage com o front-end, a interação envia uma solicitação para o back-end no formato HTTP. O back-end processa a solicitação e retorna uma resposta.

Ele faz comunicação com outra ponta, chamada frontend. Ele permite que seus usuários interajam com sua aplicação por meio de elementos visuais, como botões, caixas de seleção, gráficos e mensagens de texto. O backend consiste nos dados e na infraestrutura que fazem sua aplicação funcionar. Ele armazena e processa dados da aplicação para seus usuários.

O que é um Backend?

Quando o back-end processa uma solicitação, geralmente interage com o seguinte:

- Servidores de banco de dados para recuperar ou modificar dados relevantes;
- Microsserviços que executam um subconjunto das tarefas solicitadas pelo usuário;
- APIs (Application Program Interfaces) de terceiros para coletar informações adicionais ou executar funções adicionais.

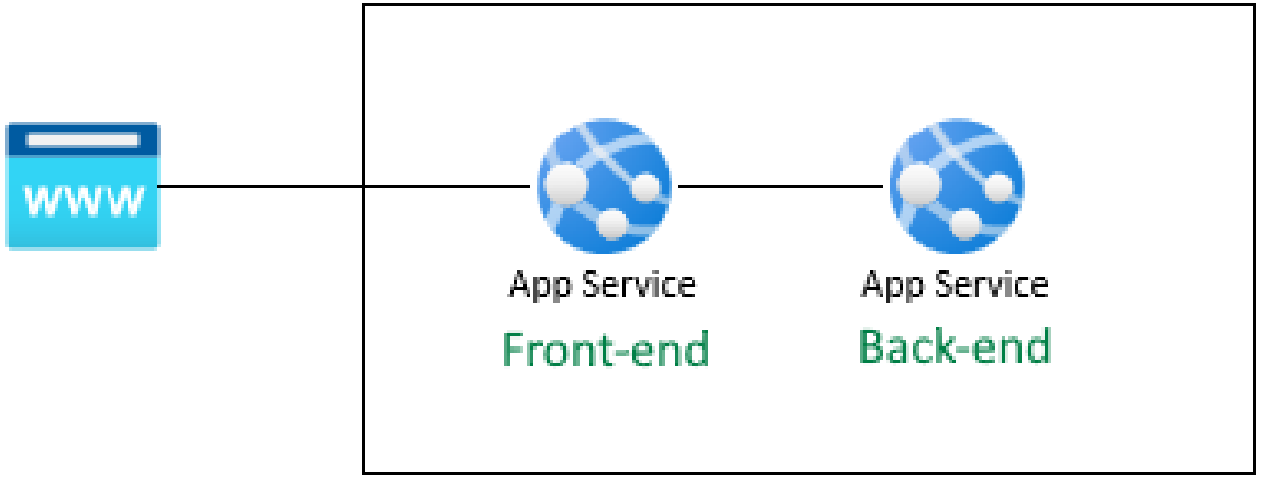
O que é um Backend?

O back-end usa vários protocolos e tecnologias de comunicação para concluir uma solicitação. Além disso, processa milhares de solicitações distintas simultaneamente. O back-end combina técnicas de simultaneidade e paralelismo, como distribuição de solicitações em vários servidores, armazenamento em cache e duplicação de dados.

Quanto mais usuários o backend deve atender, mais uso de técnicas de otimização são necessárias.

O que é um Backend?

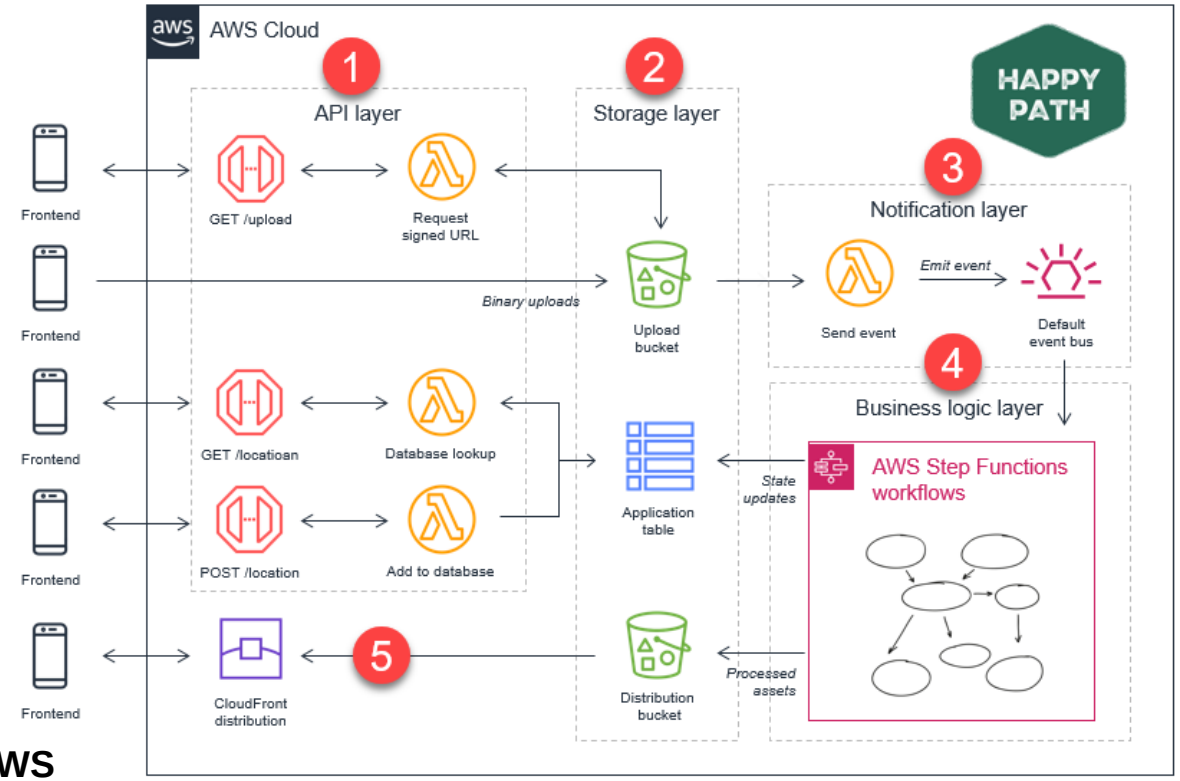
Em sistemas mais simples, podemos entender desta forma



Fonte: https://learn.microsoft.com/pt-br/azure/app-service/tutorial-auth-aad?pivots=platform-linux

O que é um Backend?

Porém pode-se tornar em algo mais complexo com várias camadas e servidores com diferentes responsabilidades.



Fonte: AWS

O que é um Backend?

Diferenças de deveres e vantagens entre frontend e backend.

Categoria	Frontend Developer	Backend Developer
Responsabilidades	**Criação da Interface do Usuário:** Desenvolve todos os componentes visuais do site ou aplicação utilizando HTML, CSS e JavaScript. Trabalha com frameworks como React, Angular e Vue.js para criar interfaces interativas.	**Gerenciamento de Banco de Dados:** Garante o armazenamento e acesso aos dados da aplicação utilizando sistemas como MySQL, PostgreSQL ou MongoDB.
	Design Responsivo: Assegura que a interface seja exibida corretamente em diferentes dispositivos (mobile, tablet, desktop), utilizando frameworks como Bootstrap.	**Programação no Lado do Servidor:** Desenvolve o código que roda no servidor utilizando linguagens como PHP, Python, Ruby, Java ou Node.js. Esse código responde às requisições do usuário e garante a funcionalidade do site.
	Melhoria da Experiência do Usuário (UX/UI): Utiliza princípios de design UX/UI para tornar a interface mais intuitiva e confortável para o usuário.	**Segurança e Otimização:** Implementa medidas de segurança e algoritmos para proteger o sistema, além de otimizar a performance para respostas rápidas e estáveis.

Fonte: Tursunbek (2024)

O que é um Backend?

Diferenças de deveres e vantagens entre frontend e backend.

Categoria	Frontend Developer	Backend Developer
Vantagens	**Evolução Rápida das Tecnologias**: Sempre surgem novas ferramentas e linguagens para aprimorar o desenvolvimento frontend.	**Estabilidade e Segurança**: Desenvolve softwares robustos e seguros.
	Resultados Visíveis Imediatamente: Alterações no código frontend podem ser visualizadas em tempo real.	**Escalabilidade**: Trabalha com grandes volumes de dados e gerencia sistemas complexos.
	Interação Direta com Usuários: Desempenha um papel crucial na melhoria da experiência do usuário.	**Oportunidade de Trabalhar com Algoritmos e Sistemas Complexos**: Desenvolve soluções avançadas para backend.

Fonte: Tursunbek (2024)

O que é um Backend?

Diferenças e semelhanças entre profissionais

Categoria	Frontend Developer	Backend Developer
Diferenças	**Campo de Atuação**: Trabalha na interface do usuário e nas páginas visíveis.	**Campo de Atuação**: Atua na funcionalidade do site, servidores e banco de dados.
	Diferenças Tecnológicas: Utiliza principalmente HTML, CSS e JavaScript.	**Diferenças Tecnológicas**: Utiliza linguagens como Python, PHP e Java para o lado do servidor.
	Responsabilidades: Foca no design da interface e na experiência do usuário.	**Responsabilidades**: Garante a segurança, velocidade e estabilidade do site.
Semelhanças	**Cooperação**: Trabalham juntos para garantir o funcionamento do sistema.	**Cooperação**: Fornece as funcionalidades necessárias para a interface do usuário.
	Resolução de Problemas: Precisam encontrar soluções criativas para problemas técnicos.	**Resolução de Problemas**: Enfrentam desafios na lógica e no desempenho do sistema.
	Testes e Otimização: Realizam testes constantes para garantir o bom funcionamento da aplicação.	**Testes e Otimização**: Implementam testes para otimizar o desempenho e a segurança do sistema.

Fonte: Tursunbek (2024)

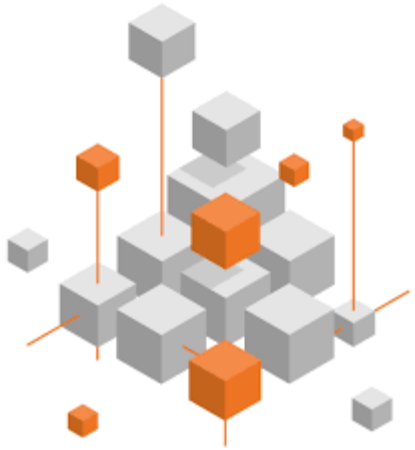
Confiabilidade, Escalabilidade, Manutenabilidade

Existem muitos fatores que podem influenciar o design de um sistema de dados, incluindo as habilidades e a experiência das pessoas envolvidas, dependências do sistema legado, o cronograma de entrega, a tolerância da sua organização a diferentes tipos de risco, restrições regulatórias, etc. Esses fatores dependem muito da situação.

Confiabilidade, Escalabilidade, Manutenabilidade

Para lidar com essa variedade de fatores, uma única ferramenta não é mais capaz de lidar com as necessidades de armazenamento e processamento de dados moderno.

Portanto, a solução é **quebrada** em pedaços menores que podem ser trabalhar eficientemente por uma **única ferramenta,** e essas ferramentas são **acopladas** juntas pelo código da aplicação.



Confiabilidade, Escalabilidade, Manutenabilidade

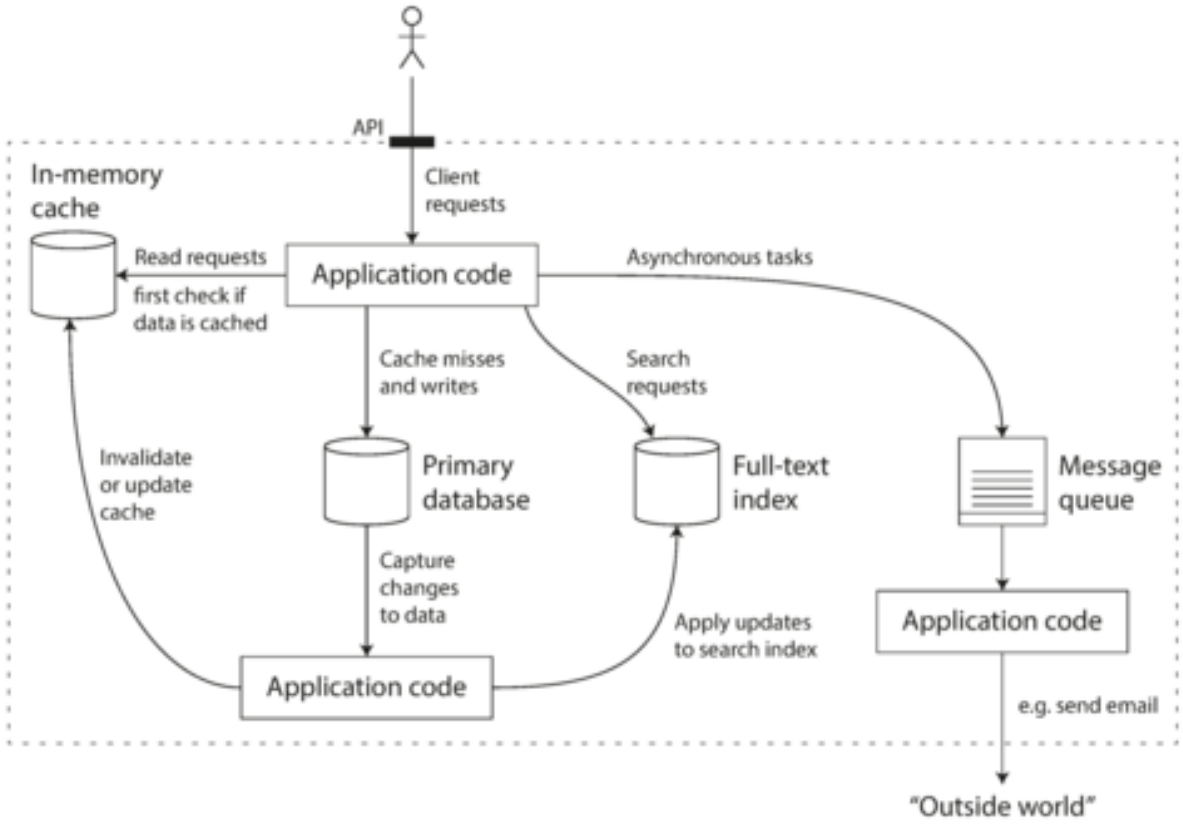


Figure 1-1. One possible architecture for a data system that combines several components.

Confiabilidade, Escalabilidade, Manutenabilidade

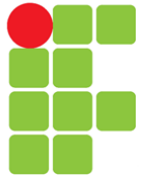
Quando você combina várias ferramentas para fornecer um serviço, a interface do serviço ou a interface de programação de aplicativos (API) geralmente oculta esses detalhes de implementação dos clientes.

Agora você criou novo sistema de dados um partir de componentes menores de propósito geral.

Agora você não é apenas um **desenvolvedor de aplicativos**, mas também um **designer de sistema de dados**. Agora temos que nos preocupas com três características desse sistema:

Confiabilidade | Escalabilidade | Manutenabilidade

Confiabilidade, Escalabilidade, Manutenabilidade



Confiabilidade

- O sistema deve continuar a funcionar corretamente (executando a função correta no nível de desempenho desejado) mesmo diante de adversidades (falhas de hardware ou software e até mesmo erro humano).

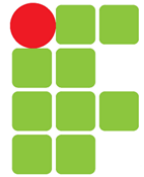
Escalabilidade

- À medida que o sistema cresce (em volume de dados, volume de tráfego ou complexidade), deve haver maneiras razoáveis de lidar com esse crescimento.

Manutenibilidade

- Com o tempo, muitas pessoas diferentes trabalharão no sistema (engenharia e operações, tanto mantendo o comportamento atual quanto adaptando o sistema a novos casos de uso), e todas elas devem ser capazes de trabalhar nele de forma produtiva.

Confiabilidade, Escalabilidade, Manutenabilidade



O que ajuda ao sistema ganhar Confiabilidade?

Lidar com falhas de hardware

- Falhas de hardware, como discos rígidos quebrando e quedas de energia, são comuns em grandes datacenters.
- Redundância de hardware (RAID, fontes duplas, geradores) reduz falhas, mas não as elimina completamente.
- Com o aumento da escala, falhas de hardware se tornam mais frequentes, exigindo soluções além da redundância física.
- Sistemas modernos priorizam tolerância a falhas via software, permitindo manutenção sem interrupções (rolling upgrades).

Lidar com erros de software

- Um bug que faz todos os servidores de aplicação falharem com uma entrada inválida, como o erro causado pelo segundo bissexto em 30 de junho de 2012 no kernel do Linux.
- Um processo descontrolado que consome recursos compartilhados, como CPU, memória, disco ou largura de banda;
- Um serviço essencial que fica lento, não responde ou retorna dados corrompidos;
- Falhas em cascata, onde um pequeno erro em um componente desencadeia falhas em outros.

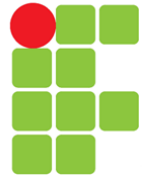
Confiabilidade, Escalabilidade, Manutenabilidade

O que ajuda ao sistema ganhar Confiabilidade?

Lidar com falhas humanas

- Modelar adequadamente as APIs para reduzir problemas de intepretação pelos programadores, encorajando a “coisa certa” a ser feita e desencorajando “a coisa errada”;
- Aplicar testes de software em todos os níveis do sistema (testes unitários, de integração e testes manuais);
- Desacoplar e isolar do sistema a parte onde há mais erros cometidos por usuários.

Confiabilidade, Escalabilidade, Manutenabilidade

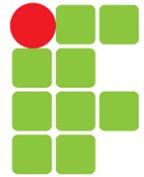


O que ajuda ao sistema ganhar Escalabilidade?

Trata-se de se questionar “Se o sistema cresce de uma maneira particular, quais são nossas opções para lidar com o crescimento?” e “Como podemos adicionar recursos de computação para lidar com a carga adicional?”

- Descrever a capacidade de carga (quantidade de dados) que o sistema suporta. A melhor escolha de parâmetros depende da arquitetura do seu sistema: podem ser solicitações por segundo para um servidor web, a proporção de leituras para gravações em um banco de dados, o número de usuários ativos simultaneamente em uma sala de bate-papo, a taxa de acertos em um cache ou algo mais.
- Quando a carga no sistema aumenta, você pode analisar como a performance muda com recursos constantes ou quanto é necessário aumentar os recursos para manter a performance. Ferramentas como Apache JMeter, Gatling, ou New Relic ajudam nesta tarefa.
- Ajustar a arquitetura do sistema para lidar com a carga medida. Pode ser obtido por *calling up* (escalonamento vertical, mudança para uma máquina mais poderosa) e *scaling out* (escalonamento horizontal, distribuição da carga em várias máquinas menores). Pode ser feita ainda por detecção automática de aumento de carga (elastic) ou manualmente (alguém deve criar novas máquinas).

Confiabilidade, Escalabilidade, Manutenabilidade



O que ajuda ao sistema ganhar Manutenabilidade?

Adicionar Operabilidade

- Oferecer visibilidade do comportamento e funcionamento do sistema, com monitoramento eficiente.
- Suporte para automação e integração com ferramentas padrão.
- Evitar dependência de máquinas individuais, permitindo manutenção sem interromper o sistema.
- Fornecer documentação clara, comportamento previsível e opções para controle manual quando necessário.

Adicionar Simplicidade

- Projetos pequenos têm código simples, mas à medida que crescem, a complexidade aumenta, dificultando a manutenção e aumentando custos.
- Sintomas de complexidade incluem acoplamento forte, dependências emaranhadas, nomes inconsistentes e **hacks** para melhorar desempenho.
- Reduzir complexidade melhora a manutenção, evitando bugs e custos excessivos; simplificar não significa reduzir funcionalidade, mas remover complexidade acidental.
- Abstrações bem-feitas ajudam a esconder detalhes de implementação, facilitando o entendimento e reutilização, embora encontrar boas abstrações seja desafiador.

Confiabilidade, Escalabilidade, Manutenabilidade

O que ajuda ao sistema ganhar Manutenabilidade?

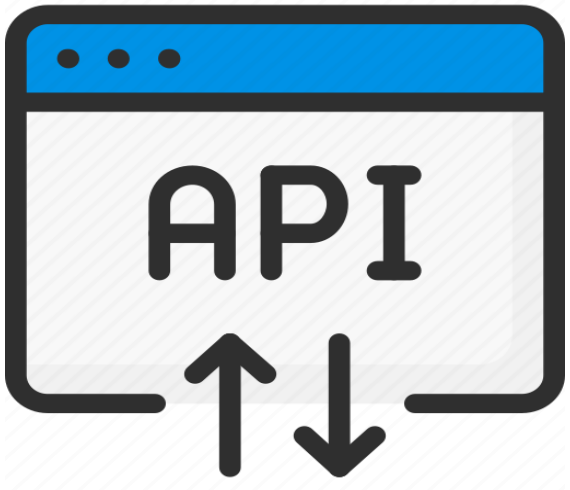
Adicionar Evolutividade

As exigências de um sistema estão em constante mudança devido a novos fatos, mudanças de prioridades, solicitações de usuários, entre outros fatores.

- O processo Ágil oferece uma estrutura para se adaptar a mudanças, com técnicas como desenvolvimento orientado a testes (TDD) e refatoração.
- A agilidade pode ser aumentada em sistemas de dados maiores, compostos por várias aplicações ou serviços, com foco em como refatorar arquiteturas complexas.
- A facilidade de modificar um sistema está diretamente ligada à sua simplicidade e abstrações, sendo mais fácil alterar sistemas simples do que os complexos.

Web API

- Uma API é a **interface** que um programa de software apresenta a outros programas, para humanos e, no caso de APIs da web, para o mundo via a Internet.
- Por meio de uma API bem especificada, o *backend* e seus diversos módulos por realizar uma comunicação coordenada para trabalharem entre si.
- APIs são os blocos de construção que permitem a **interoperabilidade** para os principais plataformas de negócios na web.



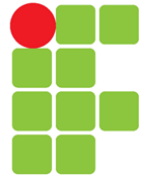
Veio da necessidade de **trocar informações** com provedores de dados que estão equipados para resolver problemas específicos, sem que outras empresas necessitem gastar tempo para resolver esse problema.

Web API

Exemplos de uso de APIs

- Criação de identidades e sua manutenção em contas de software em nuvem;
- Criação de endereços de e-mail para seu software de design colaborativo para o aplicativo da web que ajudam você a pedir entrega de pizza
- Como dados de previsão de clima são compartilhados de uma fonte confiável
- Processar seus cartões de crédito e habilitar empresas para coletar seu dinheiro perfeitamente sem se preocupar com detalhes da tecnologia financeira e suas leis e regulamentos correspondentes.

Web API



Para desenvolver uma boa API, é importante focar na sua utilidade para quem irá utilizá-la (**o cliente**). Não iremos focar nesses pontos no curso, mas podemos analisar o conselho de um profissional de TI da Uber.

> *"Uma boa API pode se resumir ao problema que você está tentando resolver e quão valioso é resolvê-lo. Você pode estar disposto a usar uma API confusa, inconsistente e mal documentada se isso significar que você está obtendo acesso a um conjunto de dados exclusivo ou funcionalidade complexa. Caso contrário, boas APIs tendem a oferecer **clareza** (de propósito, design, e contexto), **flexibilidade** (capacidade de ser adaptado a diferentes casos de uso), **poder** (completude da solução oferecida), **hackeabilidade** (capacidade de aprender rapidamente por meio de iteração e experimentação) e documentação."*
>
> —Chris Messina, developer experience lead at Uber

Web API

Padrões e Paradigmas de API

Existem diversos padrões para se implementar uma API, variando de acordo com seu objetivo. Alguns exemplos:

- RPC (Remote Procedural Call)
- REST (Representational State Transfer)
- GraphQL

Web API

RPC (Remote Procedural Call)

Forma de invocar uma “ação”, ou seja, um código/algoritmo/subrotina em um servidor remoto.

Clientes tipicamente passam um nome de método e argumentos para um servidor e recebe de volta um JSON ou XML.

As APIs RPC geralmente seguem duas regras simples:

- Os endpoints contêm o nome da operação a ser executada.
- As chamadas de API são feitas com o verbo HTTP mais apropriado: GET para solicitações somente leitura e POST para outras.

Web API

RPC (Remote Procedural Call)

Quando adotar:

- Se ações apresentam mais nuances e complexidades do que operações CRUD.
- Se as ações tiverem efeitos colaterais além daqueles sofridos pelos recursos (informações).
- Se múltiplos recursos serão afetados pelas ações ou os modelos de dados são complicados.

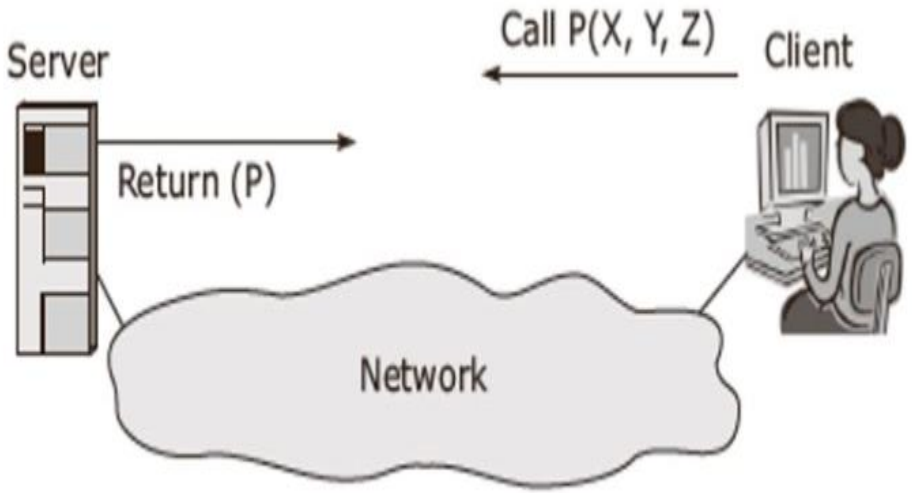


Figure 4-3 Basic RPC model

Web API

RPC (Remote Procedural Call)

Exemplos

slack api

Web API methods

</> Search methods

Popular method groups apps auth chat conversations files reactions reminders teams users usergroups views

admin.analytics.getFile	Retrieve analytics data for a given date, presented as a compressed JSON file
admin.apps.approve	Approve an app for installation on a workspace.
admin.apps.clearResolution	Clear an app resolution
admin.apps.restrict	Restrict an app for installation on a workspace.
admin.apps.uninstall	Uninstall an app from one or many workspaces, or an entire enterprise organization.

ethereum

Ethereum JSON-RPC Specification

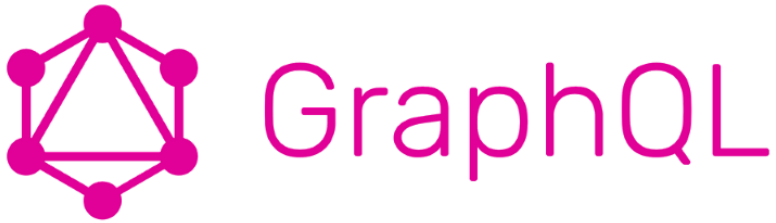
0.0.0 CC0-1.0

A specification of the standard interface for Ethereum clients.

Methods

eth_getBlockByHash	Returns information about a block by hash.
eth_getBlockByNumber	Returns information about a block by number.
eth_getBlockTransactionCountByHash	Returns the number of transactions in a block from a block matching the given block hash.
eth_getBlockTransactionCountByNumber	Returns the number of transactions in a block matching the given block number.
eth_getUncleCountByBlockHash	Returns the number of uncles in a block from a block matching the given block hash.
eth_getUncleCountByBlockNumber	Returns the number of transactions in a block matching the given block number.

Web API



GraphQL

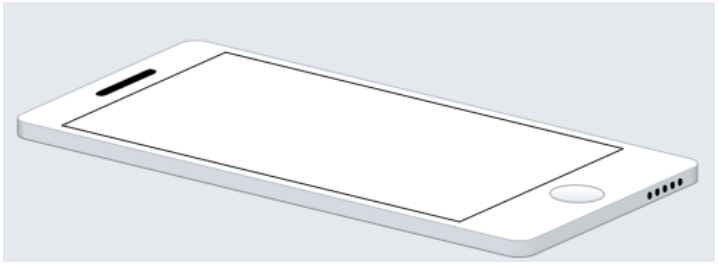
É uma linguagem de consulta (*Query Language*) para APIs que ganhou bastante popularidade recentemente.

Foi desenvolvido internamente pelo Facebook em 2012 antes de ser lançado publicamente em 2015 e foi adotado pela API provedores como GitHub, Yelp e Pinterest.

GraphQL permite que os clientes definam a estrutura dos dados necessários, e o servidor retorna essa estrutura.

Web API

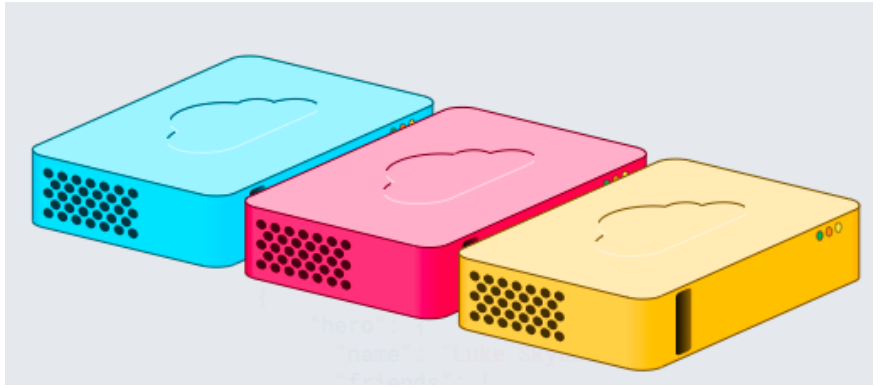
GraphQL



```
{
        user(login: "saurabhsahni") {
          id
          name
          company
          createdAt
        }
}
```

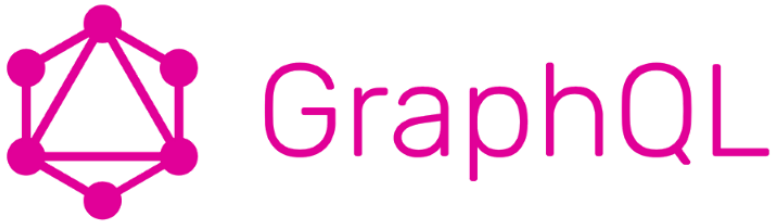
Requisição

Resposta



```
{
        "data": {
                "user": {
                        "id": "MDQ6VXNlcjY1MDI5",
                        "name": "Saurabh Sahni",
                        "company": "Slack",
                        "createdAt": "2009-03-19T21:00:06Z"
                }
        }
}
```

Web API



GraphQL

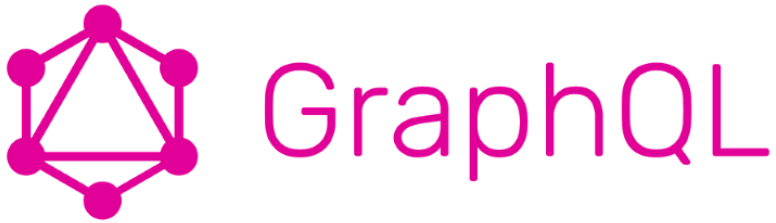
Vantagens

- O GraphQL permite que os clientes aninhem consultas e busquem dados em recursos em uma única solicitação. Com REST seriam necessárias várias.
- Você pode adicionar novos campos e tipos a uma API GraphQL sem afetar as consultas existentes. Com REST essa modificação é mais difícil.
- Com o GraphQL os clientes podem especificar exatamente o que eles precisam, os tamanhos de carga útil podem ser menores. Com REST, às vezes a resposta contém dados desnecessários.
- GraphQL é fortemente tipado, o que reduz a possibilidade de erros no cliente.
- GraphQL possui descoberta nativa. No REST precisamos utilizar alguma ferramenta, como o Swagger.

Desvantagens

- O servidor precisa fazer processamento adicional para analisar consultas complexas e verificar parâmetros.
- Otimizar o desempenho das consultas do GraphQL pode ser difícil, principalmente quando não se sabe quais consultas (querys) usuários externos irão solicitar.

Web API



GraphQL

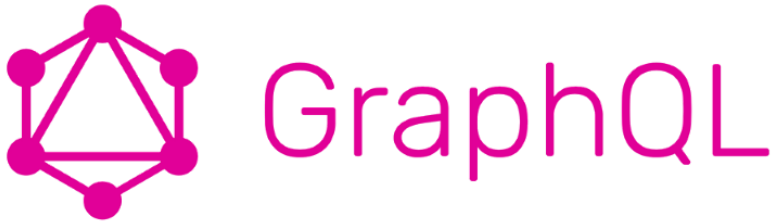
Expert Advice

One of the biggest issues GitHub saw was REST payload creep. Over time, you add additional information to a serializer for, say, a repository. It starts small but as you add additional data (maybe you've added a new feature) that primitive ends up producing more and more data until your API responses are enormous.

We've tackled that over the years by creating more endpoints, allowing you to specify you'd like the more verbose response, and by adding more and more caching. But, over time, we realized we were returning a ton of data that our integrators didn't even want. That's one of several reasons we've been investing in our GraphQL API. With GraphQL, you specify a query for just the data you want and we return just that data.

—Kyle Daigle, director of ecosystem engineering at GitHub

Web API



GraphQL

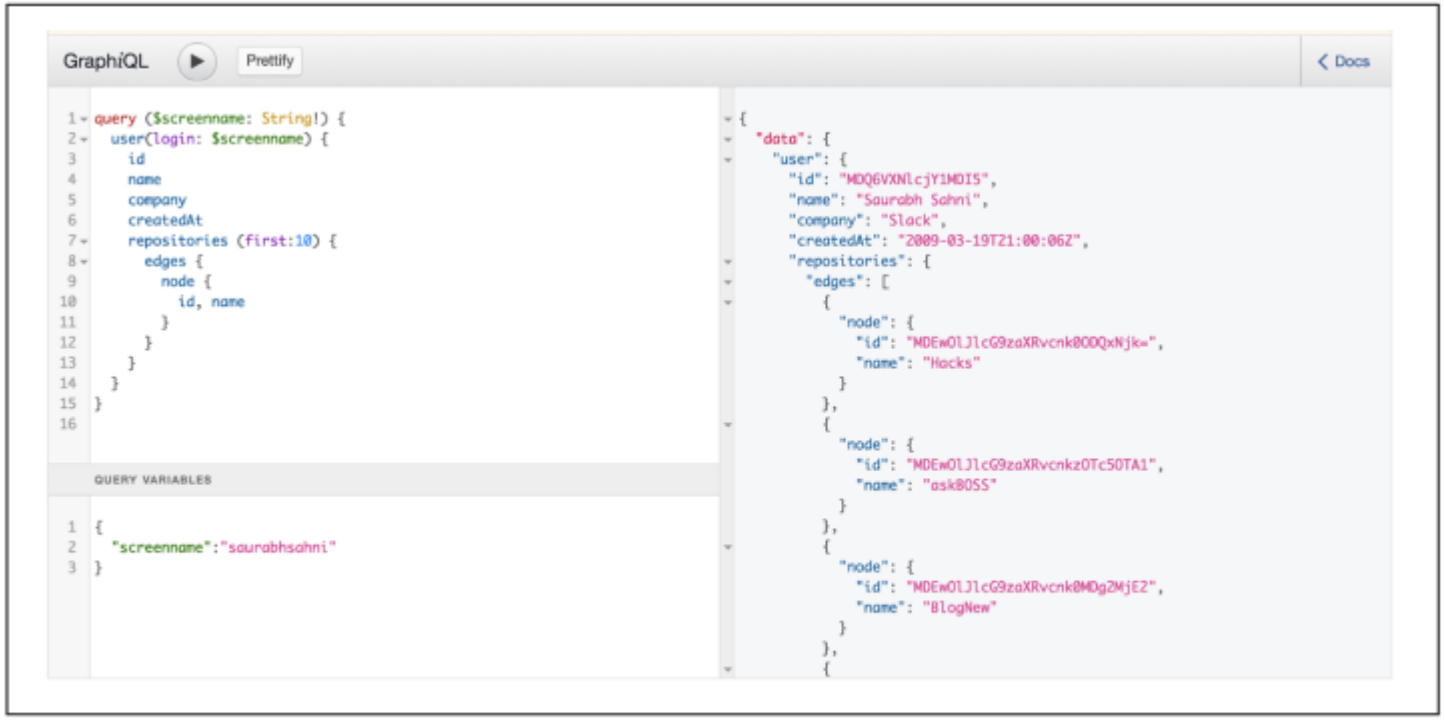


Figure 2-2. GraphiQL: GitHub's GraphQL explorer showing a complex query

REST

REST (Representational State Transfer) refere-se a um grupo de restrições de design dentro da arquitetura de software que geram sistemas distribuídos eficientes, confiáveis e escaláveis. Um sistema é denominado RESTful quando adere a todas essas restrições.

REST é uma maneira simples de organizar interações em sistemas independentes e a forma mais popular atualmente de desenvolver APIs Web.

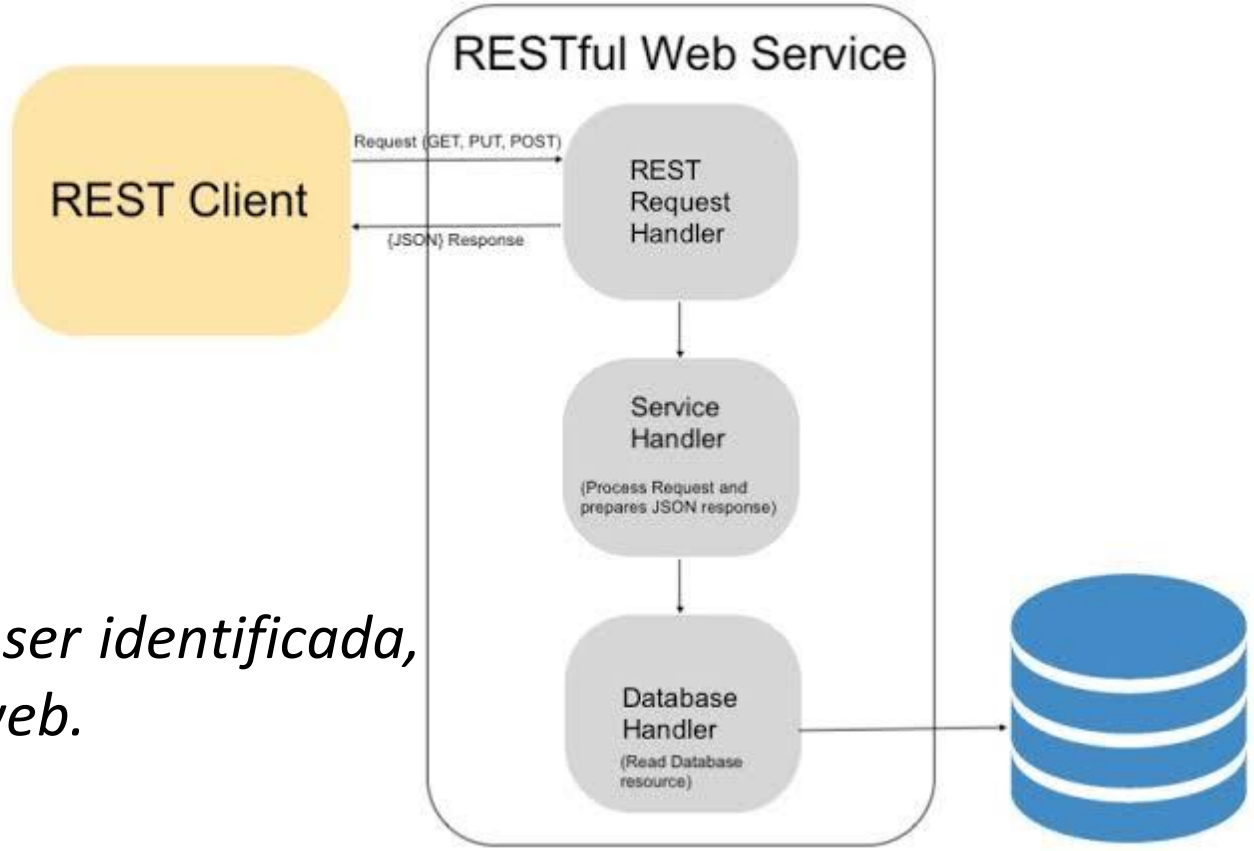
Como a implementação de REST mais conhecida para Web envolve o HTTP e o utiliza como base, foi importante primeiro entendermos como o HTTP de fato funciona antes de apresenta-lo.

REST

RESTful Applications:

Aplicações que utilizam o padrão REST para comunicação. Em uma aplicação RESTful a comunicação é realizada preferencialmente via URLs (representa um *resource*).

```
http://myserver.com/catalog/item/1729
```



Um recurso é uma entidade que pode ser identificada, nomeada, endereçada ou tratada na web.

REST

Aqui estão algumas regras gerais que as APIs REST seguem:

- Métodos HTTP como GET, POST, UPDATE e DELETE informam o servidor sobre a ação a ser executada, usualmente relacionadas a operações CRUD. Método HTTP diferente invocados no mesmo URL fornecem funcionalidades diferentes.
- Os códigos de status de resposta HTTP padrão são retornados pelo servidor indicando sucesso ou falha.
- APIs REST podem retornar respostas JSON ou XML.

REST

Aqui estão algumas regras gerais que as APIs REST seguem:

- Os recursos fazem parte de URLs, como ***`/users`***.
- Para cada recurso, duas URLs são geralmente implementadas:
  - uma para a coleção, como /users,
  - e uma para um elemento específico, como **`/users/U123`**.
- Substantivos são usados em vez de verbos para recursos.
  - Exemplo: em vez de **`/getUserInfo/U123`**, use **`/users/U123`**.

REST

Exemplo: Operações CRUD, Verbos HTTP e convenções REST

Operation	HTTP verb	URL: /users	URL: /users/U123
Create	POST	Create a new user	Not applicable
Read	GET	List all users	Retrieve user U123
Update	PUT or PATCH	Batch update users	Update user U123
Delete	DELETE	Delete all users	Delete user U123

REST

Os verbos HTTP mais importantes para criar uma API RESTful são GET, POST, PUT, e DELETE.

GET - Fornece um acesso somente leitura a um recurso.

POST - usado para criar um novo recurso.

DELETE - Usado para remover um recurso.

PUT - Usado para atualizar um recurso existente ou criar um novo recurso.

REST

Os verbos HTTP mais importantes para criar uma API RESTful são GET, POST, PUT, e DELETE.

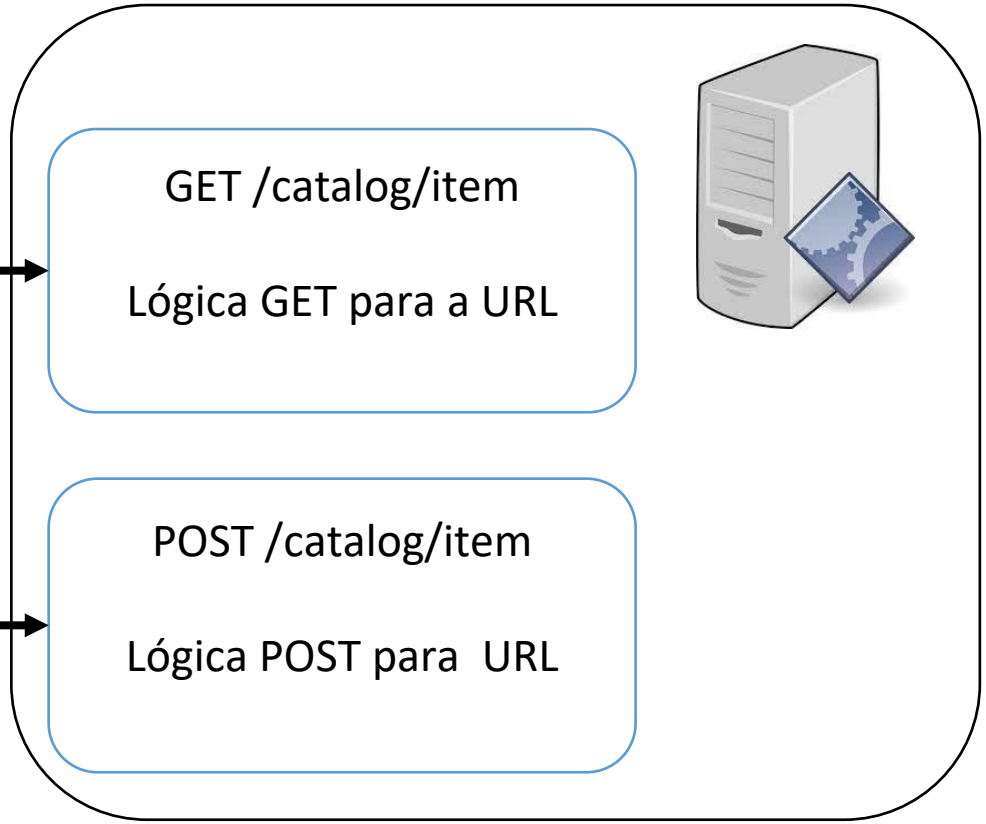
Verbo	Objetivo	Uso	Cache
GET	Recupera um novo item de um recurso	Links	Sim
POST	Cria um novo item em um recurso	Forms	Não
PUT	Substitui um itemexistente em um recursos	Forms	Não
DELETE	Deleta um item em um recurso	Forms/links	Não

REST

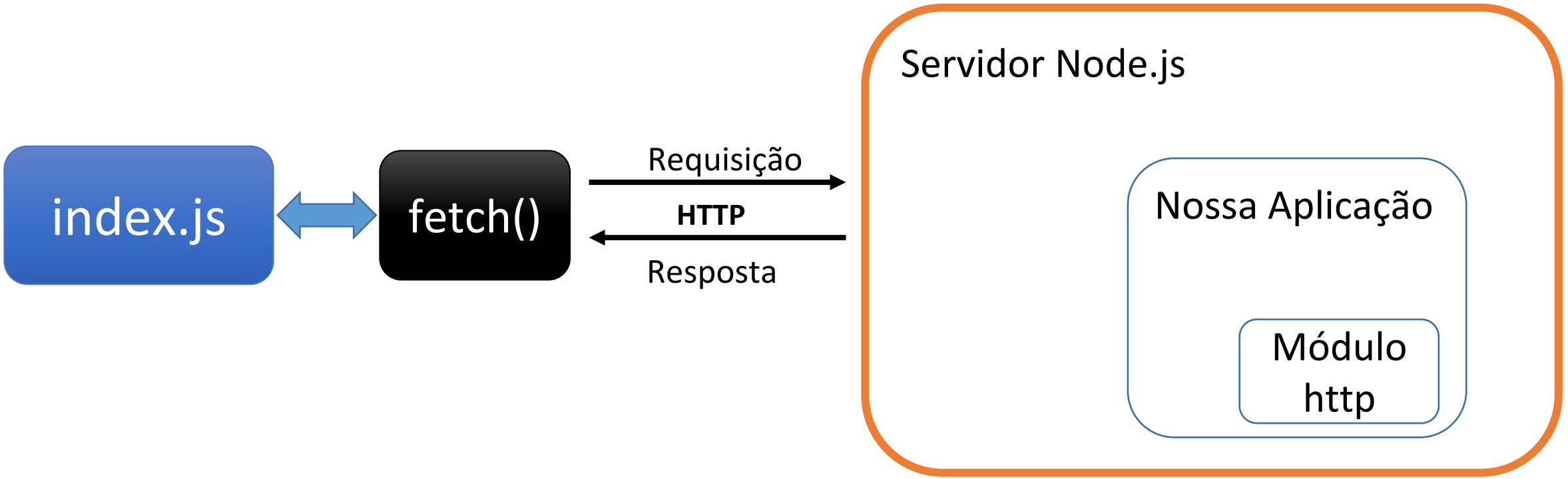
Os verbos HTTP mais importantes para criar uma API RESTful são GET, POST, PUT, e DELETE. **A combinação dos verbos com as URLs dos recursos forma identificadores únicos.**

`GET http://myserver.com/catalog/item` →

`POST http://myserver.com/catalog/item` →

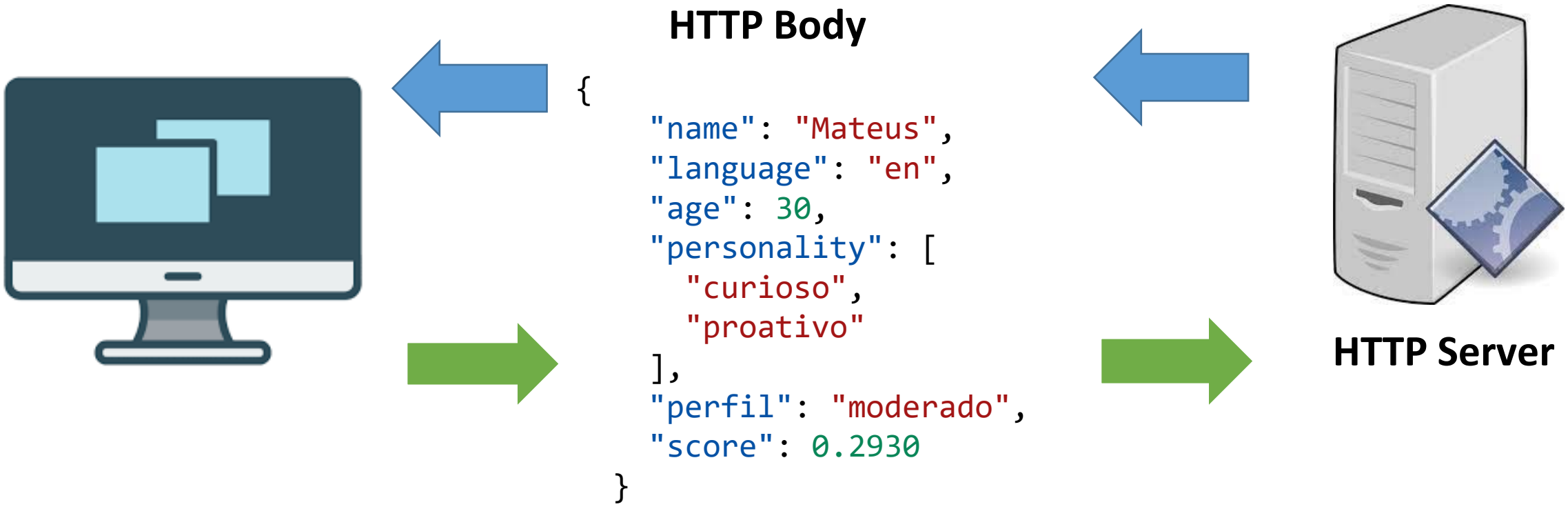


REST na arquitetura Cliente/Servidor



REST - Recebendo valores JSON

REST usa várias representações para representar um recurso como texto, JSON, XML. JSON (Javascript Object Notation) é o mais popular.



Comparação entre APIs

	REST	RPC	GraphQL
What?	Exposes data as resources and uses standard HTTP methods to represent CRUD operations	Exposes action-based API methods—clients pass method name and arguments	A query language for APIs—clients define the structure of the response
Example services	Stripe, GitHub, Twitter, Google	Slack, Flickr	Facebook, GitHub, Yelp
Example usage	`GET /users/<id>`	`GET /users.get?id=<id>`	`query ($id: String!) { user(login: $id) { name company createdAt } }`
HTTP verbs used	GET, POST, PUT, PATCH, DELETE	GET, POST	GET, POST

Comparação entre APIs

	REST	RPC	GraphQL
Pros	• Standard method name, arguments format, and status codes • Utilizes HTTP features • Easy to maintain	• Easy to understand • Lightweight payloads • High performance	• Saves multiple round trips • Avoids versioning • Smaller payload size • Strongly typed • Built-in introspection
Cons	• Big payloads • Multiple HTTP round trips	• Discovery is difficult • Limited standardization • Can lead to function explosion	• Requires additional query parsing • Backend performance optimization is difficult • Too complicated for a simple API
When to use?	For APIs doing CRUD-like operations	For APIs exposing several actions	When you need querying flexibility; great for providing querying flexibility and maintaining consistency

Web API

PADRÕES E PARADIGMAS DE API

Event-based APIs

Usados para para compartilhar dados de eventos em tempo real.

- WebHooks
- WebSockets
- HTTP Streaming

Web API

EVENT-DRIVE APIs

WebHooks

É apenas uma URL que aceita um HTTP POST (ou GET, PUT, ou DELETE).

Um provedor de API que implementa WebHooks simplesmente submete uma mensagem para a URL configurada quando algo acontecer.

Ao contrário das APIs de solicitação-resposta, com WebHooks, você pode receber atualizações em tempo real.

Web API

EVENT-DRIVE APIs

WebHooks

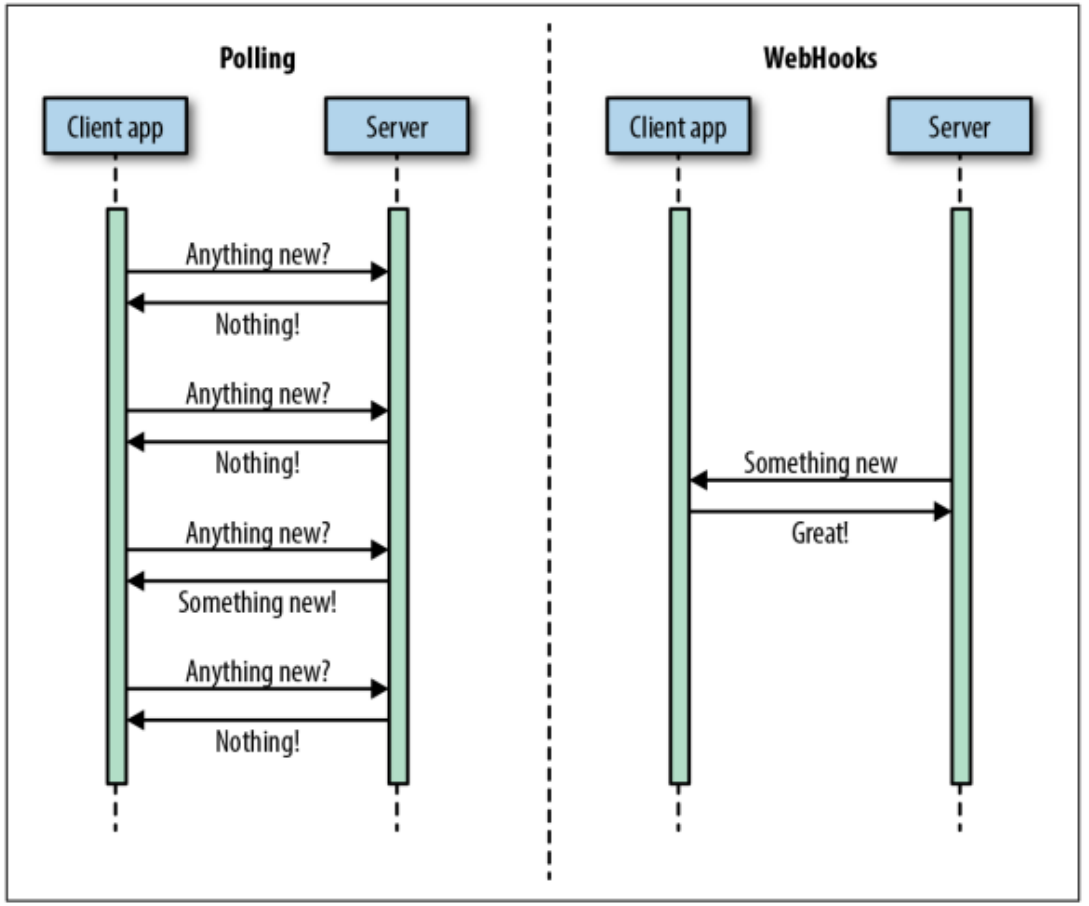


Figure 2-3. Polling versus WebHooks

Web API

EVENT-DRIVE APIs

WebSockets

É um protocolo usado para estabelecer uma comunicação de transmissão bidirecional canal de comunicação em um único protocolo TCP (Transport Control Protocol) conexão. Embora o protocolo seja geralmente usado entre uma rede cliente (por exemplo, um navegador) e um servidor, às vezes também é usado para comunicação de servidor para servidor.

Web API

EVENT-DRIVE APIs

WebSockets

Como funciona?

- O cliente pede para abrir uma conexão WebSocket e o servidor responde (se puder).
- Se esse *handshake* inicial for bem-sucedido, o cliente e o servidor concordaram em usar a conexão TCP/IP existente que foi estabelecida para a solicitação HTTP como uma conexão WebSocket.
- Os dados agora podem fluir por essa conexão usando um protocolo de mensagem.
- Uma vez que ambas as partes reconhecem que a conexão WebSocket deve ser fechada, a conexão TCP é desfeita.

Web API

EVENT-DRIVE APIs

WebSockets

⬆{"type":"tickle","id":16392}	28	18:12...
⬆{"type":"typing","channel":"C0GEV71UG","id":16393}	50	18:12...
⬆{"type":"message","channel":"C0GEV71UG","text":"new message","id":16394}	72	18:12...
⬇{"ok":true,"reply_to":16394,"ts":"1519870324.000289","text":"new message"}	74	18:12...
⬆{"type":"typing","channel":"C0GEV71UG","id":16395}	50	18:12...
⬆{"type":"message","channel":"C0GEV71UG","text":"hi","id":16396}	63	18:12...
⬇{"ok":true,"reply_to":16396,"ts":"1519870325.000226","text":"hi"}	65	18:12...
⬇{"type":"channel_marked","channel":"C0GEV71UG","ts":"1519870325.000226","unread_count":0,"unread_count_disp...	231	18:12...
⬆{"type":"typing","channel":"C0GEV71UG","id":16397}	50	18:12...
⬆{"type":"message","channel":"C0GEV71UG","text":"Hey","id":16398}	64	18:12...
⬇{"ok":true,"reply_to":16398,"ts":"1519870326.000314","text":"Hey"}	66	18:12...
⬆{"type":"typing","channel":"C0GEV71UG","id":16399}	50	18:12...
⬆{"type":"message","channel":"C0GEV71UG","text":"what's up","id":16400}	70	18:12...
⬇{"ok":true,"reply_to":16400,"ts":"1519870328.000116","text":"what's up"}	72	18:12...
⬇{"type":"channel_marked","channel":"C0GEV71UG","ts":"1519870328.000116","unread_count":0,"unread_count_disp...	231	18:12...
⬆{"type":"ping","id":16401}	26	18:12...
⬇{"type":"pong","reply_to":16401}	32	18:12...
⬇{"type":"reaction_added","user":"U0H4TC2U8","item":{"type":"message","channel":"C0GEV71UG","ts":"1519870326....	227	18:12...
⬇{"type":"pref_change","name":"emoji_use","value":"{\"train\":1,\"two\":3,\"one\":2,\"memo\":1,\"wave::skin-tone-3\":3,...	318	18:12...
⬇{"type":"reaction_added","user":"U0H4TC2U8","item":{"type":"message","channel":"C0GEV71UG","ts":"1519870326....	227	18:12...

Figure 2-5. Frames sent over a full-duplex WebSocket connection between Slack and a browser

Web API

EVENT-DRIVE APIs

HTTP Streaming

- Com as APIs de solicitação-resposta HTTP, os clientes enviam uma mensagem HTTP request e o servidor retorna uma resposta HTTP de comprimento finito.
- Agora, é possível fazer o comprimento desta resposta indeterminado. Com o HTTP Streaming, o servidor pode continuar a enviar novos dados em uma única conexão de longa duração aberta por um cliente.

Web API

EVENT-DRIVE APIs

HTTP Streaming

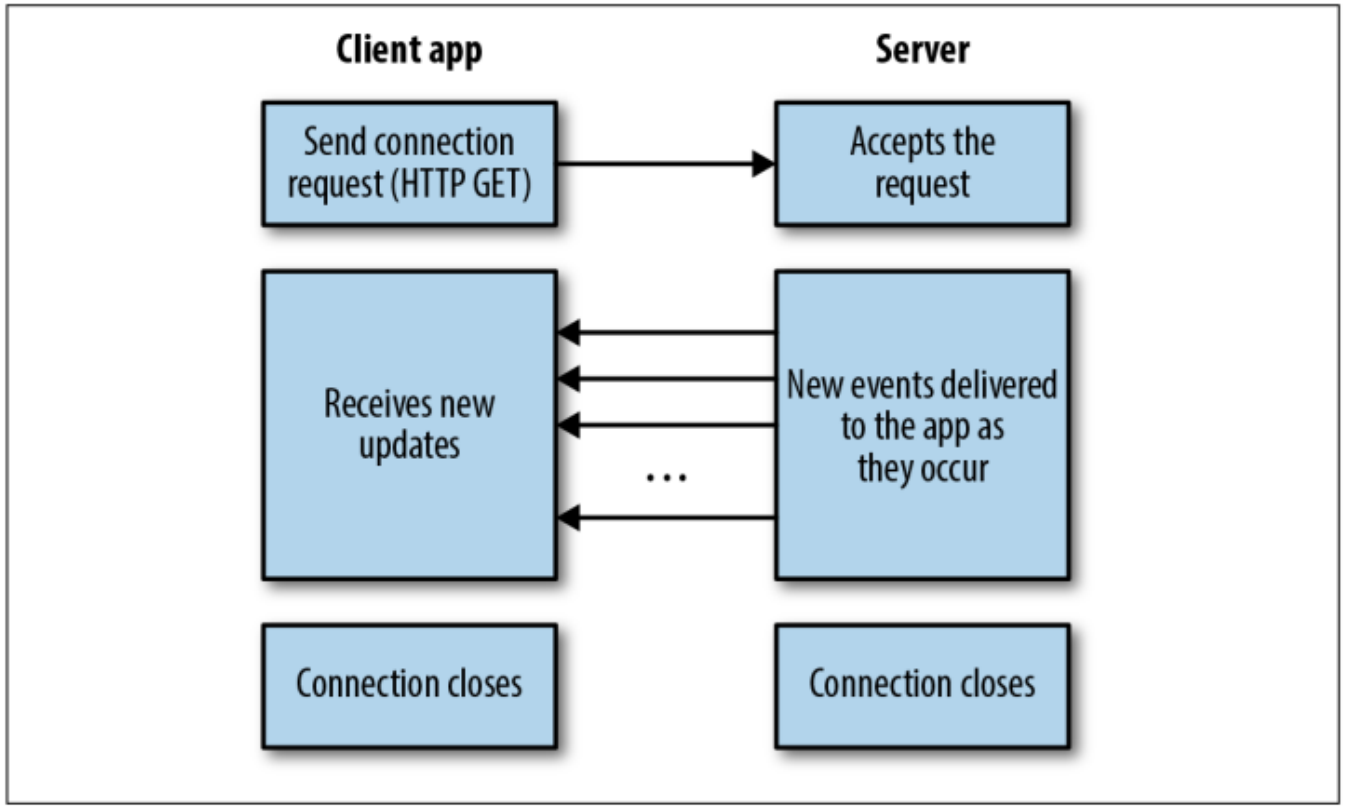


Figure 2-6. Client–server interaction with an HTTP Streaming API

Comparação entre Event-Drive APIs

	WebHooks	WebSockets	HTTP Streaming
What?	Event notification via HTTP callback	Two-way streaming connection over TCP	Long-lived connection over HTTP
Example services	Slack, Stripe, GitHub, Zapier, Google	Slack, Trello, Blockchain	Twitter, Facebook
Pros	• Easy server-to-server communication • Uses HTTP protocol	• Two-way streaming communication • Native browser support • Can bypass firewalls	• Can stream over simple HTTP • Native browser support • Can bypass firewalls
Cons	• Do not work across firewalls or in browsers • Handling failures, retries, security is hard	• Need to maintain a persistent connection • Not HTTP	• Bidirectional communication is difficult • Reconnections required to receive different events
When to use?	To trigger the server to serve real-time events	For two-way, real-time communication between browsers and servers	For one-way communication over simple HTTP

Web API

EVENT-DRIVE APIs

Exemplos

Webhook:

https://medium.com/@leandro.souara.web/criando-uma-implementa%C3%A7%C3%A3o-de-webhook-com-nodejs-4a490fb4b4c1

Websockets:

https://www.luiztools.com.br/post/como-criar-um-servidor-de-websockets-em-node-js/

HTTP Streaming

https://medium.com/@HoseungJang/video-streaming-with-node-js-9401213a04e7

https://udgwebdev.github.io/video-streaming-com-nodejs/

Fontes

Front-end x back-end — Diferença entre desenvolvimento de aplicativos — AWS. Disponível em: <https://aws.amazon.com/pt/compare/the-difference-between-frontend-and-backend>. Acesso em: 27 fev. 2025.

JALOLOV, Tursunbek. FRONTEND AND BACKEND DEVELOPER DIFFERENCE AND ADVANTAGES. **Multidisciplinary Journal of Science and Technology**, v. 4, n. 2, p. 178-179, 2024.

JIN, Brenda; SAHNI, Saurabh; SHEVAT, Amir. **Designing Web APIs: Building APIs That Developers Love**. " O'Reilly Media, Inc.", 2018.