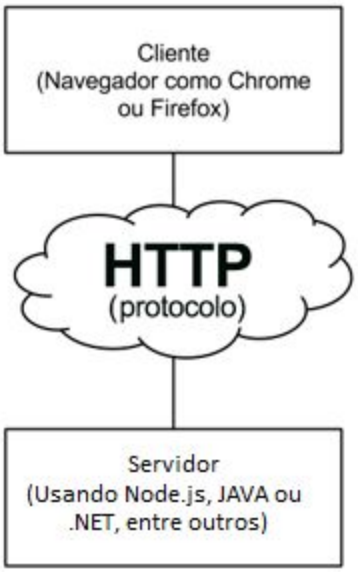
HTTP

O HTTP é um protocolo que define as regras de comunicação entre cliente e servidor na internet. Vamos focar nos próximos vídeos e entender melhor esse protocolo tão importante. mãos a obra!



API, REST e RESTFUL

API

Acrônimo de Application Programming Interface (Interface de Programação de Aplicações) é basicamente um conjunto de rotinas e padrões estabelecidos por uma aplicação, para que outras aplicações possam utilizar as funcionalidades desta aplicação.

- Responsável por estabelecer comunicação entre diferentes serviços.
- Meio de campo entre as tecnologias.

- Intermediador para troca de informações.

REST

Um acrônimo para REpresentational State Transfer (Transferência de Estado Representativo). É um conjunto de princípios de arquitetura.

Será feita a transferência de dados de uma maneira simbólica, figurativa, representativa, de maneira didática.

A transferência de dados, geralmente, usando o protocolo HTTP.

O REST delimita algumas obrigações nessas transferências de dados.

Resources seria então: Uma entidade ou um objeto.

6 NECESSIDADES (constraints) para ser RESTful

- *Uniform Interface*: Manter uma uniformidade, uma constância, um padrão na construção da interface. Nossa API precisa ser coerente para quem vai consumi-lá. Precisa fazer sentido para o cliente e não ser confusa. Logo, coisas como: o uso correto dos verbos HTTP; endpoints coerentes (todos os endpoints no plural, por exemplo); usar somente uma linguagem de comunicação (json) e não várias ao mesmo tempo; sempre enviar respostas aos clientes; são exemplos de aplicação de uma interface uniforme.

- *Client-server*: Separação do cliente e do armazenamento de dados (servidor), dessa forma, poderemos ter uma portabilidade do nosso sistema, usando o React para WEB e React Native para o smartphone, por exemplo.

- *Stateless*: Cada requisição que o cliente faz para o servidor, deverá conter todas as informações necessárias para o servidor entender e responder (RESPONSE) a requisição (REQUEST). Exemplo: A sessão do usuário deverá ser enviada em todas as requisições, para saber se aquele usuário está autenticado e apto a usar os serviços, e o servidor não pode lembrar que o cliente foi autenticado na requisição anterior. Por padrão vamos usar tokens para as comunicações.

- *Cacheable*: As respostas para uma requisição, deverão ser explicitas ao dizer se aquela resquição, pode ou não ser cacheada pelo cliente.

- *Layered System*: O cliente acessa a um endpoint, sem precisar saber da complexidade, de quais passos estão sendo necessários para o servidor responder a requisição, ou quais outras camadas o servidor estará lidando, para que a requisição seja respondida.

- *Code on demand (optional)*: Dá a possibilidade da nossa aplicação pegar códigos, como o javascript, por exemplo, e executar no cliente.

RESTFUL

RESTful, é a aplicação dos padrões REST. Em outras palavras, sistemas que utilizam os princípios REST são chamados de RESTful.

BOAS PRÁTICAS

- Utilizar verbos HTTP para nossas requisições.
- Utilizar plural ou singular na criação dos endpoints? *NÃO IMPORTA!* use um padrão!!
- Não deixar barra no final do endpoint
- Nunca deixe o cliente sem resposta!

VERBOS HTTP

- GET: Receber dados de um Resource.
- POST: Enviar dados ou informações para serem processados por um Resource.
- PUT: Atualizar dados de um Resource.
- DELETE: Deletar um Resource

STATUS CODE DAS RESPOSTAS

- 1xx: Informação

Essa resposta provisória indica que tudo está correto até agora e que o cliente deve continuar a solicitação ou ignorar a resposta se a solicitação já estiver concluída

- 2xx: Sucesso
    - 200: OK

      A solicitação foi bem sucedida. O significado do sucesso depende do método HTTP method
    - 201: CREATED

      A solicitação foi bem-sucedida e um novo recurso foi criado como resultado. Essa é geralmente a resposta enviada após solicitações POST ou algumas solicitações PUT.

- 3xx: Redirection

  A solicitação possui mais de uma resposta possível. O agente do usuário ou usuário deve escolher um deles. (Não há uma maneira padronizada de escolher uma das respostas, mas os links HTML para as possibilidades são recomendados para que o usuário possa escolher.)

- 4xx: Client Error
    - 400: Bad Request

      O servidor não conseguiu entender a solicitação devido a sintaxe inválida.

    - 401: Unauthorized

      Embora o padrão HTTP especifique "não autorizado", semanticamente essa resposta significa "não autenticado". Ou seja, o cliente deve se autenticar para obter a resposta solicitada.

    - 403: Forbidden

      O cliente não tem direitos de acesso ao conteúdo, ou seja, eles não são autorizados, portanto, o servidor está rejeitando a resposta adequada. Ao contrário de 401, a identidade do cliente é conhecida pelo servidor.

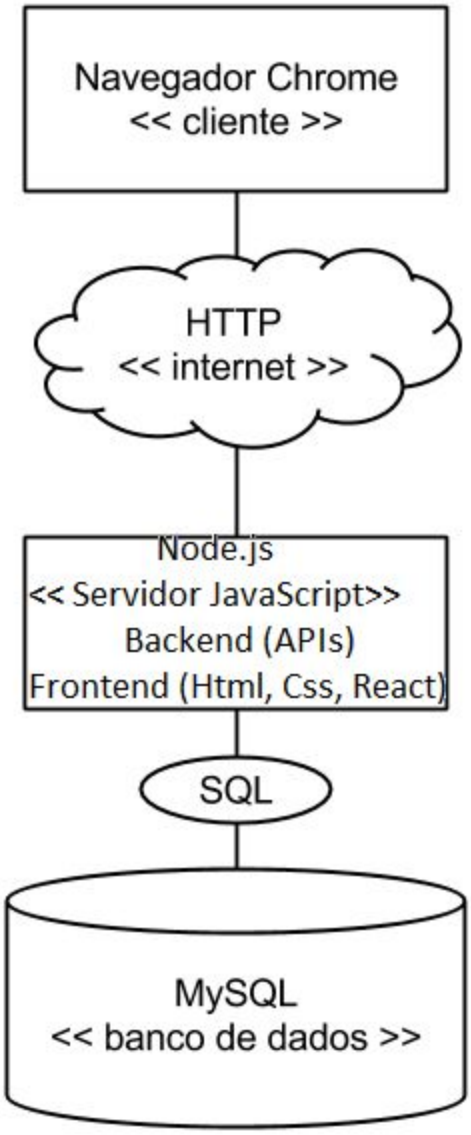
    - 404: Not Found

O servidor não consegue encontrar o recurso solicitado. No navegador, isso significa que o URL não é reconhecido. Em uma API, isso também pode significar que o terminal é válido, mas o próprio recurso não existe. Os servidores também podem enviar essa resposta em vez de 403 para ocultar a existência de um recurso de um cliente não autorizado. Esse código de resposta é provavelmente o mais famoso devido à sua ocorrência frequente na web.

- 5xx: Server Error 500: Internal Server Error

  O servidor encontrou uma situação que não sabe como lidar.

ARQUITETURA EXEMPLO


Navegador Chrome
<< cliente >>
HTTP
<< internet >>
Node.js
<< Servidor JavaScript>>
Backend (APIs)
Frontend (Html, Css, React)
SQL
MySQL
<< banco de dados >>