**HTTP**

Hypertext Transfer Protocol (HTTP) é um protocolo que permite se obter recursos na Web. É o principal responsável pela troca de dados cliente-servidor. Geralmente um navegador Web é o responsável pela reconstrução de um documento que contém diferentes sub-documentos contidos, como por exemplo texto, imagens, vídeos etc.

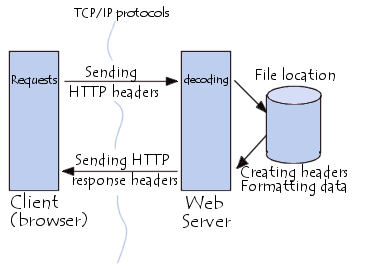


Imagem x - HTTP

Cliente e servidor se comunicam com um certo tipo de mensagem. As mensagens enviadas pelo cliente, normalmente um navegador Web, são chamadas de requests, e as mensagens enviadas como resposta pelo servidor são chamadas de responses.

O HTTP foi projetado no início dos anos 90, e evoluiu ao longo do tempo. É um protocolo de camada de aplicação enviado sobre TCP. Ele é usado além de buscar hipertexto, para buscar imagens, vídeos e pode ser usado para buscar partes de documentos para atualizar páginas da Web.

**Cliente**

O cliente é qualquer ferramenta que age em nome do usuário. Normalmente é uma função realizada pelo navegador Web, mas também pode ser realizada por programas utilizados por desenvolvedores para realizar testes, por exemplo.

O cliente é sempre a entidade que inicia as requisições, o servidor nunca é responsável por tal.

Para mostrar uma página da Web, por exemplo, o cliente envia uma requisição para buscar o HTML da página, então é realizada uma análise desse arquivo para buscar as requisições adicionais que são necessárias para a apresentação da página, como arquivos CSS, JavaScript, imagens etc. Após isso o navegador interpreta os recursos recebidos e mostra a página da Web completa.

**Servidor**

Do outro lado, temos o servidor que devolverá o que for requisitado pelo usuário. O servidor é representado por apenas uma máquina, mas na realidade pode ser uma coleção de servidores dividindo a carga para gerar todo ou parte do que foi solicitado.

**Requests**



As requisições HTTP contam com:

* Um método HTTP, geralmente GET, POST, DELETE e PUT. Normalmente as mais utilizadas são GET para pegar um recurso do servidor e POST para publicar dados de um formulário HTML, por exemplo;
* O caminho (URL) do recuso a ser buscado;
* A versão do protocolo HTTP;
* Cabeçalhos opcionais que contém informações adicionais para os servidores;
* E um corpo de dados para alguns métodos como POST, similar ao corpo das respostas.

**Responses**

****

As respostas HTTP contam com:

* A versão do protocolo HTTTP que elas seguem;
* Um código de status, indicando se foi bem sucedida ou não, e por quê;
* Uma pequena descrição informal sobre o código de status;
* Cabeçalhos HTTP, como os das requisições;
* Opcionalmente, um corpo com dados do recurso requisitado.

**API**

Application Programming Interface (API), é um conjunto de padrões e rotinas de programação para acesso a plataformas baseadas na Web.

As APIs são criadas quando uma empresa quer disponibilizar seu serviço para que os desenvolvedores de outras aplicações associem com o produto que está sendo desenvolvido. Como por exemplo a Google, que disponibiliza a API do Google Maps, que é utilizada por muitos desenvolvedores em projetos Web e em projetos Mobile.

Nos dias atuais, praticamente todo software utiliza algum tipo de API, pois é mais fácil aprender a utilizar novas APIs do que criar uma funcionalidade do zero.

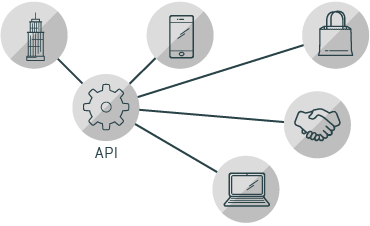


Imagem x – Representação de API

Existem, basicamente, três tipos de API:

* Privadas: são utilizadas de forma local entre aplicações de uma empresa;
* De parceiros: são utilizadas entre parceiros de negócios ou para permitir a integração entre softwares diferentes;
* Públicas: podem ser utilizadas livremente. Na maioria das vezes são disponibilizadas por empresas para que os desenvolvedores possam fazer a integração com outras aplicações.

**API REST**

Uma API REST (Representational State Transfer) utiliza requisições HTTP que são responsáveis pelas operações básicas necessárias para a manipulação dos dados.

Basicamente é um conjunto de regras utilizadas para que as requisições HTTP atendam as diretrizes definidas na arquitetura. As restrições são:

* Cliente-servidor: as aplicações existentes no servidor e no cliente devem ser separadas;
* Sem estado: as requisições são feitas de forma independente. Cada uma realiza apenas uma ação;
* Cache: a API deve utilizar cache para evitar chamadas recorrentes ao servidor;
* Interface uniforme: os recursos devem ser identificados, e sua manipulação deve ser feita por meio de representação com mensagens auto descritivas e utilizar links para navegar pela aplicação.

Quando se fala em API REST, significa utilizar uma API para acessar aplicações backend, de maneira que essa comunicação seja feita com os padrões definidos pela arquitetura REST.