**REST vs. RESTful**

REST é a arquitetura de software que utiliza os protocolos e tecnologias da web para transferir dados e realizar serviços. RESTful é o “adjetivo” dado a APIs ou sistemas que usam arquitetura REST. De acordo com a definição, RESTful seriam apenas as aplicações que utilizam de forma completa os métodos HTTP (GET, PUT, POST, DELETE, etc...) não apenas GET e POST.

**Métodos HTTP**

GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS e TRACE, são os métodos HTTP existentes, e são usados para fazer requisições do client para o server-side.  
**GET** -> Usado para solicitar dados, como o nome diz, get, pegar. Uma chamada GET costuma passar parâmetros pela URL. A requisição/request não possui um body, mas o retorno/response deve conter.  
**HEAD** -> Idêntico ao GET, porém o retorno não contem um body, somente informações de header.  
**POST** -> Envia dados para o server, o tipo de dados é definido pelo content-type. Utilizar o método POST tem como ideais: Postar dados de um form; publicar mensagens ou lista de emails; Criar novo usuário via form; Envio de dados para processamento e retorno de dados processados.   
Post não é idempotente, ou seja, postar os mesmos dados, irá criar duas requisições diferentes, e por exemplo, criar dois usuários idênticos.  
**PUT** -> Tem semelhança com o POST, exceto que o POST normalmente é usado para criar algo novo, e o PUT pode ser usado com ideia de criar OU alterar se já existente. É idempotente, se enviarmos os mesmos dados, por exemplo, para criar usuário, a primeira requisição irá criar, a segunda, alterar.  
**DELETE** -> Usado parar requisitar que o servidor delete algum recurso existente.

**HTTP e HTTPS**

Estes protocolos definem questões de segurança da navegação e transferência de dados. O HTTPS utiliza encriptação SSL, que para ser válido necessita de um certificado. Normalmente utilizam portas diferentes, o HTTP tende usar a port 80, e HTTPS a port 443. Um dominío que queira utilizar HTTPS, precisa ser validado como seguro.   
Assim, o HTTPS se difere do HTTP como mais seguro, já que requests e responses são encriptados com SSL.  
Para garantir a segurança da encriptação com HTTPS, o client tem um chave pública e o server uma chave privada, gerando assim uma chave de sessão para a conexão. Todos requests e responses vão então se comunicar com criptografia que utilize esta chave, de forma que uma tentativa de outra conexão interceptar esses requests/responses, precisaria ter a chave de sessão para descriptografar.

**CORS**

Cross-Origin Resource Sharing (CORS) é um mecanismo baseado em header-HTTP que indica quais origens (domínios, esquemas ou portas) o browser tem permissão ou não para carregar recursos.

Por exemplo, um código javascript de um domínio A quer carregar recursos de um domínio B, com um request (XmlHttpRequest). Por padrão de segurança, os browsers bloqueiam esse tipo de ação “cross-origin”, ou seja, que domínios diferentes tentem buscar/inserir recursos entre si, isso é chamado de *same-origin policy.*   
Same-origin policy indica que um domínio só pode buscar recursos em seu próprio domínio, em sua própria origem.  
Para permitir o cross-origin, é preciso que o servidor do domínio que está sendo requisitado, tenha eu seu header de resposta a configuração que permita essa ação.

O padrão Cross-Origin Resource Sharing funciona adicionando novos headers HTTP que permitem que os servidores descrevam quais origens têm permissão para ler essas informações de um navegador da web. Além disso, para métodos de solicitação HTTP que podem causar efeitos colaterais nos dados do servidor (em particular, métodos HTTP diferentes de GET ou POST com certos tipos MIME), a especificação exige que os navegadores "pré-aprovem" a solicitação, solicitando métodos suportados do servidor com o método de solicitação HTTP OPTIONS e, em seguida, mediante "aprovação" do servidor, enviando a solicitação real. Os servidores também podem informar aos clientes se "credenciais" (como Cookies e Autenticação HTTP) devem ser enviadas com solicitações.