***REST***

Representational State Transfer, abreviado como REST, não é uma tecnologia, uma biblioteca, e nem tampouco uma arquitetura, mas sim um modelo a ser utilizado para se projetar arquiteturas de software distribuído, baseadas em comunicação via rede.

REST é um dos modelos de arquitetura que foi descrito por Roy Fielding, um dos principais criadores do protocolo HTTP, em sua [tese de doutorado](http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm) e que foi adotado como o modelo a ser utilizado na evolução da arquitetura do protocolo HTTP.

Muitos desenvolvedores perceberam que também poderiam utilizar o modelo REST para a implementação de Web Services, com o objetivo de se integrar aplicações pela Web, e passaram a utilizá-lo como uma alternativa ao [SOAP](https://www.w3.org/TR/2000/NOTE-SOAP-20000508/).

REST na verdade pode ser considerado como um conjunto de princípios, que quando aplicados de maneira correta em uma aplicação, a beneficia com a arquitetura e padrões da própria Web.

REST significa Representational State Transfer. Em português, Transferência de Estado Representacional. Trata-se de uma abstração da arquitetura da Web. Resumidamente, o REST consiste em **princípios/regras/constraints**que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas. Desta forma, permitindo, por exemplo, que aplicações se comuniquem.

**E o que é RESTful?**

Muita gente, quando usam diferentes *URIs*, verbos HTTP e diferentes formatos de retorno, passam a dizer que têm uma API RESTful. Tudo isso é muito importante e faz parte, mas não é apenas isso que compõe o RESTful.

Para uma API ser considerada RESTful, ela deve seguir estritamente as regras  
definidas na arquitetura REST (com exceção da 6, obviamente, que é opcional), além de possuir um certo nível de coesão e maturidade, definidos na escala chamada de [Richardson Maturity Model](http://martinfowler.com/articles/richardsonMaturityModel.html). Falarei no próximo artigo mais detalhadamente sobre como construir na prática uma API RESTful seguindo este modelo. Mas para dar uma breve noção, os níveis são:

* **Nível 0**: Resumidamente é a ausência de qualquer regra, é apenas a  
  utilização do HTTP como transporte das operações no servidor. Normalmente se usa apenas um *endpoint* (URI) e um verbo HTTP.
* **Nível 1**: Aplicação de *resources*. A API é dividida em diferentes  
  *endpoints* que apontam para um ou mais *resources*.
* **Nível 2**: Implementação de verbos HTTP para diferentes tipos de operações  
  que deseja executar. Uma mesma URI por aceitar mais de um verbo HTTP, por exemplo: GET /user pode retornar todos os usuários e  
  POST /user passando os atributos do usuário pode criar um novo.
* **Nível 3**: Um novo conceito é adicionado, chamado:  
  [*HATEOAS*](http://en.wikipedia.org/wiki/HATEOAS) *-* (Hypertext As The Engine Of Application State). Onde a API deve fornecer para o cliente toda a informação necessária para interagir com a aplicação.

Resumo

* Nivel 0 - *HTTP*
* Nivel 1 - *HTTP + Recursos*
* Nivel 2 - *HTTP + Recursos + Verbos*
* Nivel 3 - *HTTP + Recursos + Verbos + HATEOAS (controle de hipermidias)*

As principais características ou restrições do REST são:

1. **Client-Server**
2. **Stateless**
3. **Cacheable:**
4. **Uniform Interface:**

4.1 Identification of resources: por meio das URIs nos sistemas WEB, por exemplo.  
4.2 Manipulation of resources through these representations:

4.3.Self-descriptive messages

4.4 Hypermedia as the engine of application state (HATEOAS)

1. **Layered System**
2. **Code-On-Demand (Opcional).**

A **Web Application Description Language** - Linguagem de Descrição de Aplicativo da Web (**WADL**) é uma descrição XML legível por máquina dos serviços da Web baseados em HTTP. O WADL modela os recursos fornecidos por um serviço e os relacionamentos entre eles. O WADL destina-se a simplificar a reutilização de serviços da Web baseados na arquitetura HTTP existente da Web. É independente de plataforma e idioma e visa promover a reutilização de aplicativos além do uso básico em um navegador da web. A WADL foi submetida a World Wide Web. WADL é o **equivalente REST** da Web Services Description Language (WSDL) da SOAP, que também pode ser usada para descrever serviços web REST.

**Erros HTTP**

**1xx Informativa**

**2xx Sucesso**

* Esta classe de códigos de status indica a ação solicitada pelo cliente foi recebida, compreendida, aceita e processada com êxito.

**3xx Redirecionamento**

* O cliente deve tomar medidas adicionais para completar o pedido.

**4xx Erro de cliente**

* 400 Requisição inválida (sintaxe incorreta)
* 401 Não autorizado
* 404 Não encontrado

**5xx outros erros**

* 500 Erro interno do servidor (Permissões a arquivos e pastas)
* 501 Não implementado

**Nginx** [engine x] é um servidor proxy HTTP e reverso, bem como um servidor de proxy de email, escrito por Igor Sysoev desde 2005.

O Nginx é um servidor web rápido, leve, e com inúmeras possibilidades de configuração para melhor performance.

Tecnicamente, o Nginx consome menos memória que o Apache, pois lida com requisições Web através do conceito de “**event-based web server**”; já o Apache é baseado no conceito “**process-based server**”. Eles não são necessariamente “concorrentes”, Apache e Nginx podem trabalhar juntos. É possível diminuir o consumo de memória do Apache fazendo com que as requisições Web passem primeiro pelo Nginx. Desse modo, o Apache não precisa servir arquivos estáticos, e pode depender do bom controle de cache feito pelo Nginx.

As fontes e documentação são distribuídos sob cláusula 2 licença BSD-like.

O Wikipedia utiliza Nginx como um servidor de terminação SSL, o qual é responsável por receber requisições TSL, decriptografá-las e repassá-las para outros servidores numa rede, diminuído assim a carga sobre outros servidores.

Desde a versão 5.2, o sistema operacional OpenBSD utiliza o Nginx como parte do sistema base, provendo uma alternativa ao fork do Apache 1.3 que o sistema utilizava, o qual o Nginx tinha como finalidade substituir, mas que acabou sendo subtituido por uma implementação própria de httpd.

Um **Servidor de Aplicações** (em inglês Applications Server) é um servidor que disponibiliza um ambiente para a instalação e execução de certas aplicações, centralizando e dispensando a instalação nos computadores clientes. Os servidores de aplicação também são conhecidos por middleware.

**Middleware** ou mediador, no campo da [computação distribuída](https://pt.wikipedia.org/wiki/Computa%C3%A7%C3%A3o_distribu%C3%ADda), é um [programa de computador](https://pt.wikipedia.org/wiki/Programa_de_computador) que faz a mediação entre [software](https://pt.wikipedia.org/wiki/Software) e demais aplicações. É utilizado para mor ou transportar informações e dados entre programas de diferentes [protocolos de comunicação](https://pt.wikipedia.org/wiki/Protocolos_de_comunica%C3%A7%C3%A3o), plataformas e dependências do [sistema operacional](https://pt.wikipedia.org/wiki/Sistema_operacional). É geralmente constituído por módulos dotados com **[API](https://pt.wikipedia.org/wiki/API)**s de alto nível que proporcionam a sua integração com aplicações desenvolvidas em diversas [linguagens de programação](https://pt.wikipedia.org/wiki/Linguagens_de_programa%C3%A7%C3%A3o) e **interfaces de baixo nível que permitem a sua independência** relativamente ao dispositivo. Seu objetivo é mascarar a heterogeneidade e fornecer um modelo de programação mais produtivo para os programadores de aplicativos. É composto por um conjunto de processos ou objetos em um grupo de computadores, que interagem entre si de forma a implementar comunicação e oferecer suporte para compartilhamento de recursos e aplicativos distribuídos. O Middleware é a designação genérica utilizada para referir aos sistemas de software que se executam entre as aplicações e os sistemas operacionais. O objetivo do middleware é facilitar o desenvolvimento de aplicações, tipicamente as distribuídas, assim como facilitar a integração de sistemas legados ou desenvolvidos de forma não integrada automática. Nos sistemas implementados a partir do uso de uma arquitetura de componentes distribuídos, o middleware tem a responsabilidade de gerenciar a interação entre esses componentes.

**GraphQL**

GraphQL é uma **query language para APIs** que foi [especificada](https://facebook.github.io/graphql/) pelo Facebook.

A query language do GraphQL é **fortemente tipada** e descreve, através de um schema, o modelo de dados oferecido pelo serviço. Esse schema pode ser usado para verificar se uma dada requisição é válida e, caso seja, executar as tarefas no back-end e estruturar os dados da resposta.

Um cliente pode enviar 3 tipos de requisições GraphQL, os root types:

* [query](https://facebook.github.io/graphql/#sec-Query), para consultas;
* [mutation](https://facebook.github.io/graphql/#sec-Mutation), para enviar dados;
* [subscription](https://facebook.github.io/graphql/#sec-Subscription), para comunicação baseada em eventos.

Uma API GraphQL possui apenas um endpoint e, consequentemente, só uma URL.

É possível enviar requisições GraphQL usando o método GET do HTTP, com a consulta como um parâmetro na URL. Porém, como as consultas são relativamente grandes e requisições GET tem um [limite de tamanho](https://stackoverflow.com/a/417184), o método mais utilizado pelas APIs GraphQL é o POST, com a consulta no corpo da requisição. Uma coisa é certa: com [uma consulta GraphQL](https://gist.github.com/alexandreaquiles/1367baba2e3b803ed8604950fdc039bb), eu faria menos requisições e receberia menos dados desnecessários. Mais flexibilidade e mais eficiência.

### Conceitos importantes

1. Single Endpoint: Costumam ter somente um único Endpoint, o que muda é o conteúdo do post da requisição.
2. Todas as chamadas são do tipo POST.
3. Impõem um contrato bem definido: Cliente e servidor devem respeitar uma estrutura bem definida, conhecida como Schema, para troca efetiva de dados, ou seja, é fortemente tipada.
4. Possuem operações de Query e Mutation: Query são somente leitura, não modificam os dados, substituem o GET do REST. Já Mutation são operações que alteram os dados, no caso substituem POST, PUT e DELETE.

**CORBA**

**Letra D**: “Assim como o DCOM, o CORBA é executado apenas em ambiente Windows.”

**Letra incorreta**. **DCOM** (acrônimo para Distributed Component Object Model) é uma tecnologia proprietária da **Microsoft** para criação de componentes de software distribuídos em computadores interligados em rede. O DCOM é uma extensão do COM (também da Microsoft) para a comunicação entre objetos em sistemas distribuídos.

Já o **CORBA**, da sigla Common Object Request Broker Architecture, é a especificação de um **OMA** (Object Management Architecture) que tem por objetivo a interoperabilidade entre diferentes sistemas computacionais e linguagens de programação através de ORB’s, que são estruturas que permitem que os programadores façam chamadas de um computador a outro através de uma rede. O CORBA é definido e padronizado pela OMG.

Então não, o CORBA não é executado apenas em ambiente Windows.

**Letra E**: “O modelo de referência da OMG (Object Management Group) para CORBA define a interface de aplicação, isto é, o conjunto de dados públicos do objeto que possibilita a comunicação por meio de chamadas aos métodos desse objeto com os parâmetros apropriados.”.

**Letra correta**. A **Arquitetura do Modelo de Referência da OMG** é composta por:

* Objetos de Serviços;
* Facilidades Comuns;
* Interfaces de Domínio;
* Interface de Aplicação; e
* Objetos de Aplicação.