**Arquitetura REST**

A sigla REST que significa Representational State Transfer ou traduzido para o português como transferência do estado representacional é um modelo arquitetural recente que foi introduzido e definido no ano de 2000 em uma tese PHD pelo cientista de computação norte-americano Roy T. Fielding que é também um dos principais autores da especificação do famoso protocolo HTTP.

Segundo a tese exposta por Roy Fielding a arquitetura REST é um conjunto de práticas e ideias que devem ser empregadas em uma aplicação ou software baseado na web. Tendo como objetivo padronizar as aplicações, otimizar os seus serviços, facilitar a identificação de problemas existentes nas mesmas e atrair soluções padronizadas e colaborativas.

Para alcançar estes objetivos este modelo utiliza uma serie de “regras” denominadas constraints (restrições) que sintetizam a arquitetura REST e criam abertura para o termo RESTful, o que costuma causar uma certa confusão para estudantes da área ou profissionais com pouca experiência. Enquanto REST é o modelo arquitetural até aqui descrito, RESTful é uma aplicação ou software que implementa as ideias deste modelo arquitetural e utiliza as suas constraints.

Em sequência temos as constraints descritas por Roy Fielding e uma breve explicação sobre cada uma:

**Client-Server (Cliente Servidor):**

Consiste na ideia de separar o cliente do servidor como é hoje na maioria das aplicações web, uma forma mais fácil de se perceber é pensando nos termos “front-end” e “back-end”, o fron-end representa o cliente e é a interface da aplicação com a qual o usuário ira interagir, já o back-end representa o servidor onde será feito todo o processamento dos dados e requisições vindas do front-end. Essa constraint acaba levando a vantagem de escalabilidade uma vez que o servidor pode ser melhorado ou alterado sem a necessidade de informar ao usuário ou modificar o front-end.

### Stateless (Sem Estado):

Esta constraint define que as requisições da aplicação ao servidor devem ser independentes umas das outras, fazendo com que todas as requisições tenham as informações necessárias para serem tratadas pelo servidor, de forma que o servidor não tenha conhecimento das requisições anteriores ou futuras de cada usuário individualmente. Isso acaba trazendo maior segurança para a aplicação e praticidade por se tratar cada requisição separadamente e completamente.

### Cache

O cache como em muitas outras áreas da informática se trata de uma pequena porção de memória com dados que são comumente usados por algum sistema, hardware ou software. Nessa porção de memória são colocados esses dados para que não seja necessario processa-los repetidamente durante a execução em questão, dentro da arquitetura REST podemos por exemplo podemos colocar a requisição da página inicial de uma aplicação dentro da cache melhorando assim a performance da aplicação.

### Uniform Interface (Interface Uniforme)

Nesta restrição se define que a aplicação tenha uma interface modelada entre o cliente e o servidor e que esta siga padrões importantes, levando em consideração os Recursos, Mensagem Auto Descritivas e Hypermedia.

Este conceito se assemelha bastante ao conceito de Interface de Orientação Objetos, onde temos um objeto que será nossa interface onde descrevemos os padrões nesse objeto e criamos outros objetos que implementam esse objeto, a ideia aqui é que todos os objetos que implementam a interface sigam o mesmo padrão definido na interface.

### Layered System (Sistema em Camadas)

Em um sistema em camadas deve ser possível a implementação de elementos intermediário e transparente aos clientes do sistema. Ou seja, deve-se permitir que surjam outras camadas em meios as camadas já existentes do sistema e cada camada deve ter suas interfaces de entrada e saída bem definidas e documentadas. Isso traz novamente um alto grau de escalabilidade ao sistema de forma que o cliente não venha a saber que o sistema está se adaptando e ficando cada vez mais eficiente em atender a demanda necessária e complexa.

### Code-On-Demand (Codigo sob Demanda) - Opcional

Essa se trata da única constraint considerada opcional dentro de uma arquitetura REST. Ela define que o código ou serviço de sua aplicação seja baixado no dispositivo dinamicamente respeitando a demanda necessária, assim o cliente usa somente o que realmente é necessário e requisitado pelo mesmo. Esta pratica traz maior desempenho a aplicação além de deixa-la reduzida no dispositivo, somente é opcional e não muito utilizada pelo fato de segurança onde o cliente não saberá exatamente o que está sendo baixado o que pode causar certo estranhamento. Além de tornar o sistema levemente mais suscetível a explorações ou invasões indesejadas.

**Protocolo JSON**

O significado de JSON é JavasSript Object Notation ou em português Notação de Objeto JavaScript se trata de um formato de texto leve para troca de dados entre sistemas derivado do JavaScript, este formato além de leve é também simples de ser lido por nós humanos e não apenas interpretado por programas como em outros protocolos. O JSON se utiliza de solicitações assíncronas e tem a sua própria extensão para arquivo que é .json.

O JSON também é relativamente novo sendo especificado originalmente por Douglas Crockford no início dos anos 2000, foi Douglas Crockford quem popularizou o JSON como uma alternativa ao XML que era comumente utilizado na época. E desde então o JSON vem sendo cada vez mais utilizado e adotado pelos mais variados sistemas que buscam suas características práticas, que se enquadram muito bem no ambiente de sistemas distribuídos na web que temos atualmente.

Como especificado por Douglas Crockford o JSON pode representar 4 tipos primitivos de informações: strings, números, booleans e null. Toda informação contida em um JSON é agrupada por pares, estes pares são sempre respectivamente: O nome do item e a informação contida nele. Os objetos são separados uns dos outros por chaves “{}” como se fossem blocos e as informações dentro desses blocos são separadas por virgulas.

Exemplo de um Objeto JSON para melhor compreensão:

{

"Latitude": 37.7668,

"Longitude": -122.3959,

"City": "SAN FRANCISCO",

"State": "CA",

"Zip": "94107",

"Country": "US"

}

Como apontado pelo Engenheiro de Software Sênior e Instrutor da Caelum Alexandre Gama o JSON traz consigo diversas vantagens em relação ao seu antecessor XML (que ainda é usado). Essas vantagens são: compreensão por humanos já citada, a fácil conversão de seus tipos de dados, uma melhora significativa na performance de transporte de dados, o tamanho do arquivo reduzido e como mencionado a pouco o JSON retorna objetos do tipo JSON que podem ser facilmente convertidos para objetos de linguagens orientadas a objetos, simplificando e facilitando muito a codificação, tratamento e processamento dos dados.

Além de todos os benefícios e praticidades já citadas o JSON também é uma tendência crescente entre as maiores empresas de desenvolvimento do mundo em aplicações como: Google, Facebook, Twitter, Yahoo!, etc... Isso pode ser levado como um forte de indicio de que essa tecnologia receberá um bom suporte e permanecerá funcional por pelo menos alguns anos no mercado.

# Referências

Roy Thomas Fielding (em inglês)

<<http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm> > Acesso em 17 de junho de 2017.

Udemy. **Online Courses** - Jackson Pires (Engenheiro de Software, Professor acadêmico)

<<https://www.udemy.com/restful-apis/learn/v4/content> > Acesso em 18 de junho de 2017.

Douglas Crockford (especificador do JSON, em inglês)

< <http://www.ietf.org/rfc/rfc4627.txt?number=4627> > Acesso em 18 de junho de 2017.

Devmedia. **Codigos para quem faz códigos.** Alexandre Gama (Engenheiro de Software Sênior e Instrutor da Caelum) <<http://www.devmedia.com.br/introducao-json/23166>> Acesso em 18 de junho de 2017.