APIs REST

Prof.Ademir Mazer Jr
@nunomazer

API

- API - **A**pplication **P**rogramming **I**nterface (Interface de Programação de Aplicações) é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não necessitam envolver-se em detalhes da implementação, e sim apenas usar (consumir) seus serviços.
- Podem ser as interfaces de uma biblioteca ou framework de programação, por exemplo, a API da linguagem PHP ou do framework Laravel; interfaces de um programa disponível localmente, por exemplo, a API da suíte LibreOffice; ou a mais comumente citada atualmente, interface de comunicação via Web para usar serviços diversos, por exemplo, a API Google, Twitter, Facebook, SalesForce, etc

Web Service

- Serviço da Web (WS):
  - um serviço oferecido por um dispositivo eletrônico para outro dispositivo eletrônico, comunicando-se entre si através da World Wide Web, ou
  - um servidor em execução em um dispositivo de computador, ouvindo solicitações em uma porta específica em uma rede, servindo documentos da web (HTML, JSON, XML, imagens) e criando serviços de aplicativos da web, que servem na resolução de problemas de domínio específicos sobre a Web (WWW, Internet, HTTP)
- Em um WS, uma tecnologia da Web, como o protocolo HTTP, é usada para transferir formatos de arquivo legíveis por máquina, como XML e JSON.

REST

- REST - **RE**presentational **S**tate **T**ransfer (Transferência Representacional de Estado): é um estilo arquitetônico para sistemas hipermídia distribuídos, apresentado pela primeira vez por Roy Fielding em 2000 em sua dissertação.
- Como qualquer outro estilo de arquitetura, o REST também tem suas próprias restrições de orientação, em número de 6, que devem ser satisfeitas se uma interface precisar ser definida como *RESTful*.

Princípios Orientadores do REST

- **1. Cliente/Servidor** (Client-Server)
- Ao separar as questões da interface do usuário das questões de armazenamento de dados, melhoramos a portabilidade da interface do usuário em várias plataformas e melhoramos a escalabilidade simplificando os componentes do servidor.

Princípios Orientadores do REST

- **2. Sem estado** (Stateless)
- cada solicitação do cliente para o servidor deve conter todas as informações necessárias para entender a solicitação e não pode tirar proveito de qualquer contexto armazenado no servidor. O estado da sessão é, portanto, mantido inteiramente no cliente.

Princípios Orientadores do REST

- **3. Armazenável em cache** (Caheable)
- as restrições de cache exigem que os dados em uma resposta a uma solicitação sejam implicitamente ou explicitamente rotulados como armazenáveis ou não armazenáveis em cache. Se uma resposta puder ser armazenada em cache, o cache do cliente terá o direito de reutilizar os dados de resposta para solicitações equivalentes posteriormente.

Princípios Orientadores do REST

- **4. Interface uniforme** (Uniform interface)
- ao aplicar o princípio de generalidade da Engenharia de Software à interface do componente, a arquitetura geral do sistema é simplificada e a visibilidade das interações é aprimorada. Para obter uma interface uniforme, várias restrições arquitetônicas são necessárias para orientar o comportamento dos componentes. REST é definido por quatro restrições de interface: identificação de recursos; manipulação de recursos por meio de representações; mensagens autodescritivas; e hipermídia como o motor de estado do aplicativo.

Princípios Orientadores do REST

- **5. Sistema em camadas** (Layered system)
- O estilo de sistema em camadas permite que uma arquitetura seja composta de camadas hierárquicas, restringindo o comportamento dos componentes de forma que cada componente não possa "ver" além da camada imediata com a qual está interagindo.

Princípios Orientadores do REST

- **6. Código sob demanda** (Code on demand) **opcional*
- REST permite que a funcionalidade do cliente seja estendida por meio do download e da execução de código na forma de miniaplicativos ou scripts. Isso simplifica os clientes, reduzindo o número de recursos necessários para serem pré-implementados.

Recurso

- A abstração chave das informações no REST são os recursos.
- Qualquer informação que pode ser nomeada pode ser um recurso: um documento ou imagem, um serviço temporário, uma coleção de outros recursos, um objeto não virtual (por exemplo, uma pessoa) e assim por diante.
- REST usa um identificador de recurso para identificar o recurso particular envolvido em uma interação entre componentes.

Recurso

- O estado do recurso em qualquer *timestamp (*carimbo de data/hora) específico é conhecido como representação de recurso.
- Uma representação consiste em dados, metadados que descrevem os dados e links hipermídia que podem ajudar os clientes na transição para o próximo estado desejado.

Recurso

- O formato de dados de uma representação é conhecido como *media type* (tipo de mídia).
- O tipo de mídia identifica uma especificação que define como uma representação deve ser processada.
- Uma API verdadeiramente RESTful parece hipertexto.
  - Cada unidade endereçável de informação carrega um endereço, seja explicitamente (por exemplo, atributos de link e id) ou implicitamente (por exemplo, derivado da definição do tipo de mídia e estrutura de representação).

Recurso – segundo Roy Fielding

- “Hipertexto (ou hipermídia) significa a apresentação simultânea de informações e controles de tal forma que a informação se torna o recurso através do qual o usuário (ou autômato) obtém escolhas e seleciona ações. Lembre-se de que o hipertexto não precisa ser HTML (ou XML ou JSON) em um navegador. As máquinas podem seguir links quando entendem o formato dos dados e os tipos de relacionamento.”
  - Além disso, as representações de recursos devem ser autodescritivas: o cliente não precisa saber se um recurso é um funcionário ou dispositivo. Ele deve agir com base no tipo de mídia associado ao recurso. Portanto, na prática, você acabará criando muitos tipos de mídia personalizados - normalmente um tipo de mídia associado a um recurso.
- “Cada tipo de mídia define um modelo de processamento padrão. Por exemplo, o HTML define um processo de renderização para hipertexto e o comportamento do navegador em torno de cada elemento. Não tem relação com os métodos de recurso GET / PUT / POST / DELETE / ... além do fato de que alguns elementos de tipo de mídia definirão um modelo de processo que funciona como, por exemplo: /elementos âncora, com um atributo href, criam um link de hipertexto que, quando selecionado, invoca uma solicitação de recuperação (GET) no URI correspondente ao atributo href codificado por CDATA/. ”

Métodos de Recursos

- Ainda associado ao REST estão os métodos de recursos a serem usados para realizar a transição desejada. Um grande número de pessoas relaciona incorretamente os métodos de recursos aos métodos (verbos) HTTP: GET / PUT / POST / DELETE.
- Roy Fielding nunca mencionou qualquer recomendação sobre qual método ser usado em qual condição. Tudo o que ele enfatiza é que deve existir uma interface uniforme. Se você decidir que o HTTP POST será usado para atualizar um recurso - em vez de a maioria das pessoas recomendar o HTTP PUT - está tudo bem e a interface do aplicativo será RESTful.
- De maneira ideal, tudo o que é necessário para alterar o estado do recurso deve fazer parte da resposta da API para esse recurso - incluindo métodos e em que estado eles deixarão a representação após sua chamada.

Métodos de Recursos – segundo Roy Fielding

- Uma API REST deve ser inserida sem nenhum conhecimento prévio além do URI inicial (marcador) e conjunto de tipos de mídia padronizados que são apropriados para o público-alvo (ou seja, deve ser entendida por qualquer cliente que possa usar a API).
- A partir desse ponto, todas as transições de estado do aplicativo devem ser conduzidas pela seleção do cliente de opções fornecidas pelo servidor que estão presentes nas representações recebidas ou implícitas pela manipulação do usuário dessas representações.
- As transições podem ser determinadas (ou limitadas) pelo conhecimento do cliente de tipos de mídia e mecanismos de comunicação de recursos, ambos os quais podem ser melhorados em tempo real (por exemplo, código sob demanda). [A falha aqui implica que a informação fora do escopo está conduzindo a interação em vez de hipertexto.]
  - Outra coisa que o ajudará na construção de APIs RESTful é que os resultados da API baseada em consulta devem ser representados por uma lista de links com informações resumidas, não por matrizes de representações de recursos originais porque a consulta não substitui a identificação de recursos.

REST e HTTP não são a mesma coisa

- Muitas pessoas preferem comparar HTTP com REST. REST e HTTP não são iguais.
- Porém, como o REST também pretende tornar a web (internet) mais simples e padronizada, ele defende o uso dos princípios do REST de forma mais estrita. E neste ponto que as pessoas tendem a comparar REST com web (HTTP). Roy Fielding, em sua dissertação, em nenhum lugar mencionou qualquer diretiva de implementação - incluindo qualquer preferência de protocolo e HTTP.
- Isto significa que, se você está aderente aos 6 princípios orientadores do REST, você pode chamar sua API de RESTful.

REST e HTTP não são a mesma coisa

- Em outras palavras, no estilo de arquitetura REST, os dados e funcionalidade são considerados recursos e são acessados usando Uniform Resource Identifiers (URIs).
- Os recursos são acionados por meio de um conjunto de operações simples e bem definidas.
- Os clientes e servidores trocam representações de recursos usando uma interface e protocolo padronizados - normalmente HTTP.

REST e HTTP não são a mesma coisa

- Os recursos são separados de sua representação para que seu conteúdo possa ser acessado em uma variedade de formatos, como HTML, XML, texto simples, PDF, JPEG, JSON e outros.
- Os metadados sobre o recurso estão disponíveis e são usados, por exemplo, para controlar o cache, detectar erros de transmissão, negociar o formato de representação apropriado e executar autenticação ou controle de acesso. E o mais importante, toda interação com um recurso não mantém estado.
- **Todos esses princípios ajudam os aplicativos RESTful a serem simples, leves e rápidos.**

REST e HTTP não são a mesma coisa

- Em outras palavras, no estilo de arquitetura REST, os dados e funcionalidade são considerados recursos e são acessados usando Uniform Resource Identifiers (URIs).
- Os recursos são acionados por meio de um conjunto de operações simples e bem definidas.
- Os clientes e servidores trocam representações de recursos usando uma interface e protocolo padronizados - normalmente HTTP.