REpresentational State Transfer API

Leonardo Anjoletto Ferreira

REST

- Arquitetura para sistemas de hipermídia distribuídos
- Mais usado para APIs web
- Estilo híbrido derivado de outras arquiteturas para redes
- A tese de doutorado completa propondo o REST pode ser encontrada em https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Elementos da arquitetura REST

- Ignora detalhes de implementação e sintaxe de protocolo
- Foco em:
  - funções dos componentes
  - restrições nas interações com outros componentes
  - interpretação dos elementos de dados

Elementos dos dados

Apenas 3 opções para distribuição de hipermídia:

1. renderização no local de armazenamento e /envio do dado processado
2. encapsulamento e envio do dado e ferramenta para renderização
3. envio do dado bruto e metadado com processamento pelo cliente

- REST permite um formato híbrido dos 3...

Elementos dos dados em REST

- Compartilhamento da descrição dos tipos dos dados usando metadados
- Limitação do escopo do que pode ser informado pela interface padrão
- Componentes se comunicam enviando uma representação do dado usando um formato padrão, baseado no que o cliente pode/consegue/deseja fazer
- Separação de responsabilidades + escalabilidade + obscurecimento de informações

Recurso

- Qualquer coisa
- Mapeamento conceitual
- Qualquer informação que pode ser nomeada é um recurso
- Qualquer conceito que pode ser referenciado em um hipertexto pode ser definido como um recurso
- Ou seja, qualquer coisa
- E pode ser temporário / mudar com o tempo

Representação

- Sequência de bytes + metadados que descrevem os bytes (+ metadados que descrevem os metadados)
- Componentes REST realizam chamadas a recursos usando uma representação que informam o estado atual ou o estado desejado
- Metadados no formato de pares chave-valor
- Respostas podem contar tanto dados + metadados como metadados sobre o recurso usado

Conectores

- Um conector é uma interface abstrata para um componente de comunicação
- Por ser abstrato, é possível trocar a implementação do conector sem que o usuário perceba
- Por gerenciar uma comunicação de rede, as informações podem vir de mais do que um recurso
- Possui vários tipos de conectores para encapsular recursos e enviar representações
- Stateless

Componentes

- Servidor de origem: servidor que recebe as requisições
- Gateway (reverse proxy): intermediário usado pela rede ou servidor de origem para encapsular a interface de outros serviços
- Proxy: intermediário selecionado pelo cliente que encapsula serviços
- User agent: que inicia a requisição e recebe a resposta final

Princípios/restrições do REST

Objetivo de promover simplicidade, escalabilidade e statelessness, usa 6 princípios como guia:

1. Interface uniforme
2. Modelo cliente-servidor
3. Stateless
4. Pode ser armazenado em cache (Cacheable)
5. Sistema em camadas
6. Code on Demand

REST - Interface uniforme

- **Identificação de recursos:** cada interface deve identificar de forma única uma interação entre cliente e servidor
- **Manipulação de recursos a partir de representação:** os recursos devem possuir uma representação uniforme para a resposta do servidor

REST - Interface uniforme

- **Mensagens auto descritivas:** cada recurso deve possuir informações sobre o processamento da mensagem e também informações sobre ações adicionais que o cliente pode executar
- **Hipermídia como meio de estado da aplicação:** cliente deve possuir apenas a URI inicial e toda interação adicional deve ser feita por hyperlinks

REST - Modelo cliente-servidor

- Cliente: interface do usuário
- Servidor: armazenamento e processamento de dados
- O padrão cliente-servidor obriga a aplicação de separação de responsabilidades e permite que cada um seja desenvolvido de forma independente.
- Facilita a portabilidade do código para novas interface
- Facilita a escalabilidade dos componentes do servidor
- **Devemos garantir que as regras de negócio não quebrem**

REST - Stateless

- O servidor não armazena estados de interações anteriores
- Cada mensagem trocada entre cliente e servidor deve conter toda a informação necessária para realizar a requisição
- Portanto, o cliente deve manter todo estado (dados) da aplicação

REST - Cacheable

- A resposta de uma requisição deve identificar se é possível ser mantida em cache ou não
- Se a resposta for mantida em cache (cacheable), o cliente pode reutilizar o dado para requisições equivalentes dentro de um período específico

REST - Sistema em camadas

- Model View Controller (MVC)?
- Divisão da arquitetura em camadas permite a separação de comportamentos de clientes
- Cada componente interage apenas com as camadas imediatamente

REST - Code on Demand

- **Opcional**
- O servidor pode enviar código para ser executado pelo cliente
- Clientes podem ser mais simples e estendidos
- Reduz visibilidade pois a funcionalidade não está descrita

Visualização da arquitetura

- Visão do processo: exibe o relacionamento das interações dos Componentes, permitindo visualizar a passagem dos dados por todo o sistema
- Visão do conector: exibe a mecânica da comunicação entre os componentes
- Visão dos dados: exibe o estado da aplicação conforme os dados passam pelos componentes da aplicação

O que não falamos?

- Métodos HTTP (POST, PUT, GET, DELETE, etc)
- JavaScript Object Notation (JSON)