
Fecomércio RS
Senac

Desenvolvimento de Serviços e APIs

Aula09

Arquitetura REST

Prof. MSc. Adalto Selau Sparremberger

assparremberger@senacrs.com.br

@adaltoss

/assparremberger

Fecomércio RS | Senac

API – Application Programming Interface

- Interface de Programação de Aplicações
- Conjunto de rotinas e padrões estabelecidos por uma aplicação, para que outras aplicações possam utilizar as funcionalidades desta aplicação.
- Estabelece comunicação entre diferentes serviços/aplicações.

REST
Representational State Transfer

- Modelo de arquitetura de software distribuído, baseadas em comunicação via rede.
- Roy Fielding, um dos criadores do protocolo HTTP, descreveu REST em sua tese de doutorado.
- É um modelo para projetar arquiteturas de software distribuídos, baseadas em comunicação em rede.

Requisitos REST

- Utiliza geralmente o protocolo HTTP para a transferência de dados
- Para a construção de uma API, REST delimita 6 constraints (obrigações ou requisitos)

Fonte: https://github.com/Rocketseat/youtube-api-rest-restful

Requisitos REST

- **Uniform Interface**: Manter uma uniformidade, uma constância, um padrão na construção da interface. Nossa API precisa ser coerente para quem vai consumi-la. Precisa fazer sentido para o cliente e não ser confusa. Logo, coisas como: o uso correto dos verbos HTTP; endpoints coerentes (todos os endpoints no plural, por exemplo); usar somente uma linguagem de comunicação (json) e não várias ao mesmo tempo; sempre enviar respostas aos clientes; são exemplos de aplicação de uma interface uniforme.
- **Client-server**: Separação do cliente e do armazenamento de dados (servidor), dessa forma, poderemos ter uma portabilidade do nosso sistema, usando o React para WEB e React Native para o smartphone, por exemplo.

Fonte: https://github.com/Rocketseat/youtube-api-rest-restful

Requisitos REST

- **Stateless**: Cada requisição que o cliente faz para o servidor, deverá conter todas as informações necessárias para o servidor entender e responder (RESPONSE) a requisição (REQUEST). Exemplo: A sessão do usuário deverá ser enviada em todas as requisições, para saber se aquele usuário está autenticado e apto a usar os serviços, e o servidor não pode lembrar que o cliente foi autenticado na requisição anterior.
- **Cacheable**: As respostas para uma requisição, deverão ser explicitas ao dizer se aquela requisição, pode ou não ser cacheada pelo cliente.
- **Layered System**: O cliente acessa a um endpoint, sem precisar saber da complexidade, de quais passos estão sendo necessários para o servidor responder a requisição, ou quais outras camadas o servidor estará lidando, para que a requisição seja respondida.
- **Code on demand** (optional): Dá a possibilidade da nossa aplicação pegar códigos, como o Javascript, por exemplo, e executar no cliente.

Fonte: https://github.com/Rocketseat/youtube-api-rest-restful

RESTful

Apostila: https://blog.caelum.com.br/rest-principios-e-boas-praticas/

- Estar RESTful é cumprir os requisitos (necessidades) REST,
- RESTful, é a aplicação dos padrões REST.

Dicas de Boas Práticas

- Utilizar verbos HTTP para nossas requisições.
- Utilizar plural ou singular na criação dos endpoints?

  NÃO IMPORTA! use um padrão!!
- Não deixar barra no final do endpoint
- Nunca deixe o cliente sem resposta!

Fonte: https://github.com/Rocketseat/youtube-api-rest-restful

VERBOS HTTP

- **GET**: Receber dados de um Resource.
- **POST**: Enviar dados ou informações para serem processados por um Resource.
- **PUT**: Atualizar dados de um Resource.
- **DELETE**: Deletar um Resource
- **PATCH**: Atualizar parcialmente um determinado recurso.
- **HEAD**: Similar ao GET, mas utilizado apenas para se obter os cabeçalhos de resposta, sem os dados em si.
- **OPTIONS**: Obter quais manipulações podem ser realizadas em um determinado recurso.

Fonte: https://github.com/Rocketseat/youtube-api-rest-restful

STATUS DAS RESPOSTAS

- **1xx:** Informação
- **2xx:** Sucesso
  - **200:** OK
  - **201:** CREATED
  - **204:** Não tem conteúdo PUT POST DELETE
- **3xx:** Redirection
- **4xx:** Client Error
  - **400:** Bad Request
  - **404:** Not Found!
- **5xx:** Server Error
  - **500:** Internal Server Error

Fonte: https://github.com/Rocketseat/youtube-api-rest-restful

Construindo uma API REST com Node.JS – Preparando o projeto

- Crie uma pasta em qualquer lugar do seu sistema. EX: **/api**
- Dentro da pasta **/api** execute o comando **npm init** que solicitará alguns dados sobre o seu projeto. Após informarmos todos os dados, este processo criará o arquivo **package.json** e uma pasta chamada **node_modules** , que contém todas as bibliotecas padrão do NodeJS.

package.json

```
{
  "name": "api-rest",
  "version": "1.0.0",
  "description": "Minha API REST",
  "main": "index.js",
  "scripts": {
    "test": "echo 'Erro: nenhum teste especificado ' && exit 1"
  }
  "keywords": [
    "API",
    "REST",
    "NodeJS"
  ],
  "autor": "<Seu Nome>",
  "license": "ISC",
  "dependências": {
    "analisador de corpo": "^ 1.18.2",
    "express": "^ 4.16.2"
  }
}
```

Construindo uma API REST com Node.JS – Preparando o projeto

- Agora vamos instalar os módulos NPM que precisaremos, execute os comandos:
  - **npm install –save express**
  - **npm install –save body-parser**
- O Express é o módulo mais popular para desenvolvimento Web no NodeJS e também é especialmente útil para a construção de APIs.
- O Body-parser, que anteriormente fazia parte do Express, nos ajudará a processar a carga útil (ou corpo) da solicitação, que nos deixará prontos e no formato que precisamos, neste caso, em JSON.

Construindo uma API REST com Node.JS – Construindo um servidor

- Montar um servidor usando o NodeJS + Express, para o qual, teremos que criar um novo arquivo chamado **server.js** diretamente na raiz do projeto

```
1   var express = require('express');
2   var http = require('http');
3   var app = express();
4
5   app.get('/', (req, res) => {
6     res.status(200).send("Bem-vindo ao API REST");
7   });
8
9   http.createServer(app).listen(8001, () => {
10    console.log('Servidor iniciado em http://localhost:8001');
11  });
```

Construindo uma API REST com Node.JS – Construindo um servidor

- Na linha 1 importamos o módulo Express e na linha 2 módulo http , que faz parte do padrão NodeJS, portanto, ele precisa ser instalado com o NPM. Este segundo módulo é usado para criar ouvintes capazes de ouvir solicitações HTTP em uma determinada porta. O segundo passo será criar uma instância do Express, como podemos ver na linha 3.

```
1   var express = require('express');
2   var http = require('http');
3   var app = express();
4
5   app.get('/', (req, res) => {
6     res.status(200).send("Bem-vindo ao API REST");
7   });
8
9   http.createServer(app).listen(8001, () => {
10    console.log('Servidor iniciado em http://localhost:8001');
11  });
```

Construindo uma API REST com Node.JS – Construindo um servidor

- Depois de construída a instância do Express, precisamos criar apenas os roteadores, que são as regras pelas quais o Express saberá como processar as diferentes solicitações em diferentes URLs, para as quais o Express fornecerá uma série de funções para processar o diferentes métodos que o HTTP suporta, como GET , POST , DELETE , PUT etc.
- Nesse caso, criamos um roteador para atender às solicitações que atingem a raiz do servidor. É por isso que a importância da linha 5, que responderá com a frase "Bem-vindo ao API REST" ao receber uma solicitação GET na raiz do servidor. Assim como temos a função GET, temos todas as outras, por exemplo, POST, DELETE, PUSH, PUT etc. que são definidos da mesma maneira que GET, mas é necessário alterar a função pelo método HTTP desejado, por exemplo: app.post() , app.delete(), app.push() , app.put( ).

Construindo uma API REST com Node.JS – Construindo um servidor

- Todas essas variantes recebem dois parâmetros, o primeiro é o URL ao qual eles respondem e o segundo é uma função (retorno de chamada) que será executada assim que uma solicitação for recebida.
- O servidor Http que criamos (createServer) passando como parâmetro, a instância do Express para finalmente subir o servidor com a função de ouvinte . O último recebe a porta na qual escuta e um retorno de chamada que é executado depois que o servidor é gerado.

Referências

- https://github.com/Rocketseat/youtube-api-rest-restful
- https://www.oscarblancarteblog.com/2018/01/15/construir-api-rest-nodejs-tercera-parte/


Fecomércio RS
Senac