

**MBA
USP
ESALQ**

MICROSSERVIÇOS

Prof. José Maria Cesário Jr.

*A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.

Proibida a reprodução, total ou parcial, sem autorização. Lei nº 9610/98

EXERCÍCIO

Aula 01 - Simulação de REST APIs

Objetivos

1. Conhecer uma ferramenta para simulação de REST APIs
2. Simular o uso de uma REST API
3. Testar os métodos HTTP e rotas em uma aplicação simulada
4. Entender os principais conceitos de manipulação de uma REST API

Ferramentas

1. cUrl
2. Docker
3. JSON-Server
4. Linha de comando do sistema operacional
5. Navegador web

cUrl

- cURL é uma ferramenta de linha de comando utilizada para obter ou enviar dados, incluindo arquivos, usando a sintaxe URL.
- Disponível para Sistemas Operacionais baseados em UNIX/LINUX, Windows, MacOS
- Mais informações: <https://curl.se/>

Instalação cUrl

- GNU/Linux, distribuições baseadas em Debian (como o Ubuntu), execute: `sudo apt install curl`
- Windows: pré-instalado, mas caso não estiver, acesse: <https://curl.se/windows/>
- Distribuições baseadas em Arch Linux, execute: `sudo pacman -S curl`
- MacOS: pré-instalado, mas caso não estiver, execute: `brew install curl` (necessita instalar Brew previamente)

Fonte: Wikipedia ▢ <https://pt.wikipedia.org/wiki/CURL>

Métodos HTTP

Método HTTP	Descrição
GET	Lê/Consulta um recurso
POST	Cria um recurso
PUT	Atualiza/Altera um recurso
PATCH	Atualiza/Altera um recurso parcialmente
DELETE	Remove um recurso

O método de requisição HTTP PUT cria um novo recurso ou substitui uma representação do recurso de destino com os novos dados.

A diferença entre PUT e POST é que PUT é idempotente: chamá-lo uma ou várias vezes sucessivamente terá o mesmo efeito (não é um efeito colateral), enquanto usar POST repetidamente pode ter efeitos adicionais, como executar uma ação várias vezes.

Fonte: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods/PUT>

Comandos cUrl relacionados com REST

Comando cUrl	Descrição	Exemplo
-i ou --include	Inclui os cabeçalhos de resposta na resposta	curl -i http://www.exemplo.com
-d ou --data	Inclui dados para postar na URL. Os dados precisam ser codificados por url	curl -d "data-to-post" http://www.exemplo.com
-H ou --header	Envia o cabeçalho da solicitação para o recurso. Os cabeçalhos são comuns com solicitações de API REST	curl -H "key:12345" http://www.exemplo.com
-X POST	Especifica o método HTTP a ser usado com a solicitação (neste exemplo, POST). Se você usar -d na solicitação, curl especificará automaticamente um método POST. Com solicitações GET, incluir o método HTTP é opcional, porque GET é o método padrão usado.	curl -X POST -d "resource-to-update" http://www.exemplo.com

JSON Server

- JSON Server é uma aplicação que simula REST APIs com base em arquivos JSON simples
- <https://github.com/typicode/json-server>
- <https://www.npmjs.com/package/json-server>

Criar a imagem Docker

1. Criar uma pasta chamada app1
2. Criar o arquivo Dockerfile dentro da pasta app1

```
app1 > Dockerfile > ...  
1 FROM node:latest  
2  
3 RUN npm install -g json-server  
4  
5 RUN echo '{"carros":[{"id":1,"marca":"gm","modelo":"corsa"}, {"id":2,"marca":"ford","model":"fiesta"}]}' > /tmp/base.json  
6  
7 ENTRYPOINT ["json-server", "--port", "8080", "--host", "0.0.0.0"]  
8  
9 CMD ["/tmp/base.json"]
```

Construir (build) e executar o container

1. Criar o container (build)

```
docker build -t jsonserver .
```

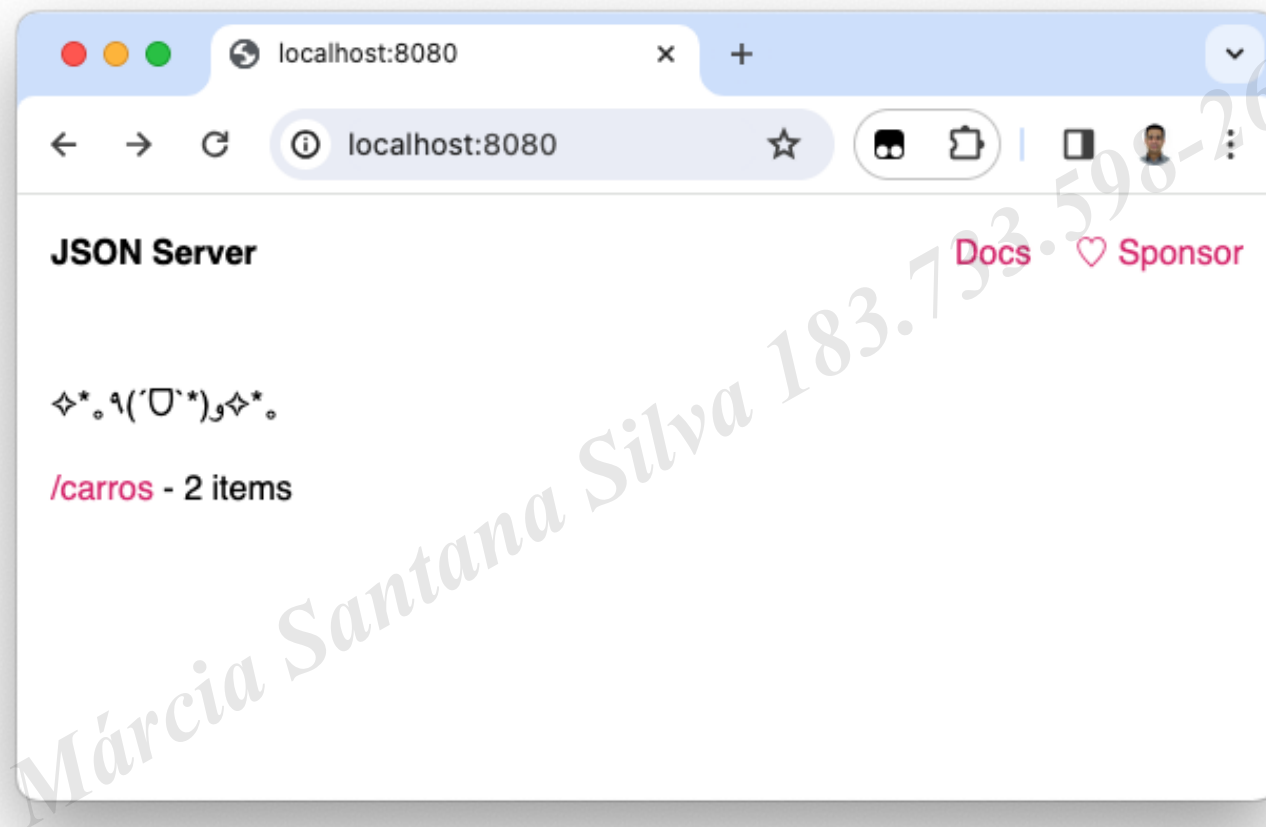
2. Executar o container criado

```
docker run -d --rm -it --name jsonserver-container -p 8080:8080 jsonserver
```

3. Via navegador, acessar o endereço <http://localhost:8080/>

- *utilize 127.0.0.1, caso localhost não funcione*

Resultado



<http://localhost:8080/>

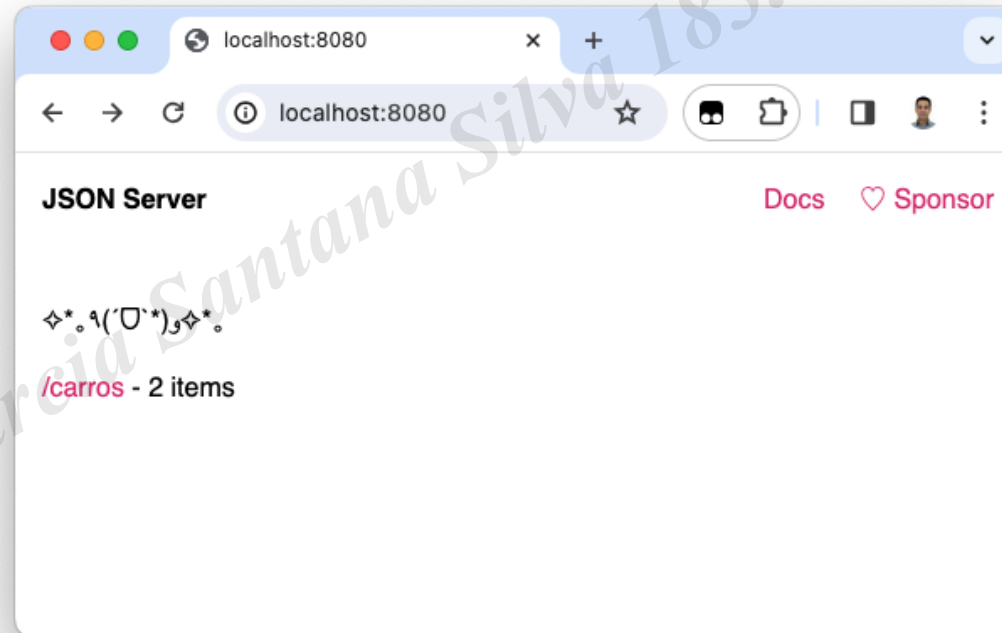
- *utilize 127.0.0.1, caso localhost não funcione*

Linha de comando do Sistema Operacional

- No seu sistema operacional, abra o prompt da linha de comando para seguir com os exercícios
- Digite os comandos conforme o exercício
- Você também pode copiar os comandos, prestando atenção para exatidão dos mesmos e/ou cópia de partes não corretas

JSON-Server

- Acompanhe em uma janela separada do seu navegador as atualizações no endpoint, conforme você executa os comandos



`http://localhost:8080/`

- utilize 127.0.0.1, caso localhost não funcione

Exercício 1

- Testar o método HTTP GET, via comando curl (método padrão)

- LINUX/macOS

`curl http://0.0.0.0:8080/carros` ou `curl -X GET http://0.0.0.0:8080/carros`

- Windows (CMD) Resultado esperado

`curl http://localhost:8080/carros` ou `curl -X GET http://localhost:8080/carros`

```
[
  {
    "id": "1",
    "marca": "gm",
    "modelo": "corsa"
  },
  {
    "id": "2",
    "marca": "ford",
    "model": "fiesta"
  }
]
```

- utilize 127.0.0.1, caso localhost ou 0.0.0.0 não funcione

Exercício 2

- Carregar mais dados através do arquivo carros.json
- Na pasta app1, verificar se o arquivo carros.json está disponível
- Digitar o comando

```
docker cp carros.json jsonserver-container:tmp/base.json
```

- Resultado esperado

```
Successfully copied 2.56kB to jsonserver-container:tmp/base.json
```

Exercício 2

- Listar o container atual

```
docker ps
```

- Resultado esperado

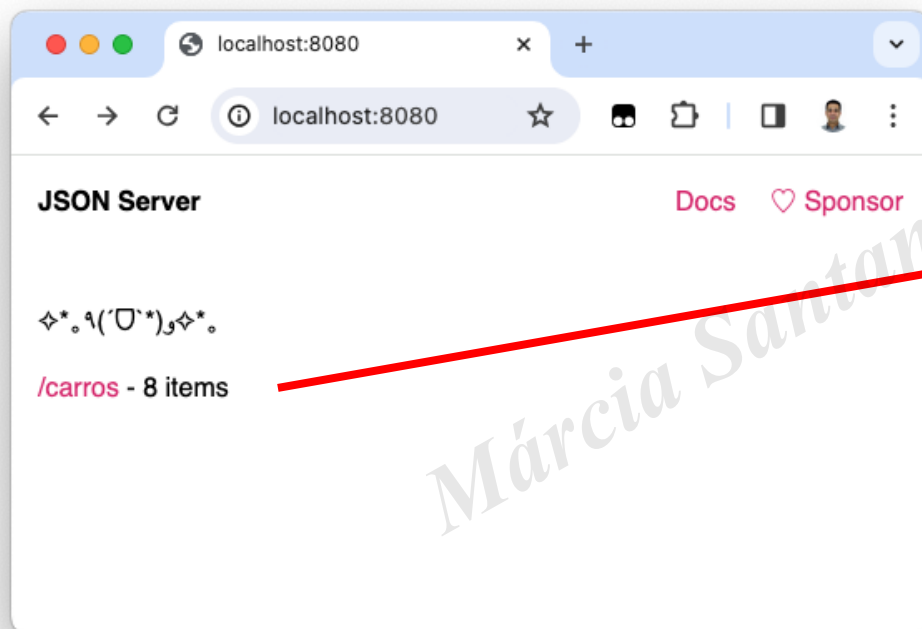
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
a5849de9225e	jsonserver	"json-server --port ..."	About an hour ago	Up 35 minutes	0.0.0.0:8080->8080/tcp

- Reiniciar o container atual (atenção para copiar o seu container ID)

```
docker restart a5849de9225e
```

Exercício 2

- Resultado esperado JSON-server



- utilize 127.0.0.1, caso localhost não funcione

Exercício 3

- Testar chamada do método HTTP POST via linha de comando

- LINUX/macOS

```
curl -i -H "Accept: application/json" -H "Content-type: application/json" -X POST http://0.0.0.0:8080/carros -d '{"id": "9", "marca": "audi", "modelo": "q7"}'
```

- WINDOWS (CMD)

```
curl -i -H "Accept: application/json" -H "Content-type: application/json" -X POST http://localhost:8080/carros -d '{"id\\":\\"9\\",\\"marca\\":\\"audi\\",\\"modelo\\":\\"q7\\"}'
```

- Resultado esperado

```
HTTP/1.1 201 Created
X-Powered-By: tinyhttp
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, HEAD, PUT, PATCH, POST, DELETE
Access-Control-Allow-Headers: content-type
Content-Type: application/json
Date: Tue, 30 Jan 2024 02:14:23 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 52

{
  "id": "9",
  "marca": "audi",
  "modelo": "q7"
```

- utilize 127.0.0.1, caso localhost ou 0.0.0.0 não funcione

Exercício 4

- Testar chamada do método HTTP PUT via linha de comando

- LINUX/macOS

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X PUT http://0.0.0.0:8080/carros/9 -d '{"id": "9", "marca": "bmw", "modelo": "x1"}'
```

- WINDOWS (CMD)

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X PUT http://localhost:8080/carros/9 -d '{"id": "9", "marca": "bmw", "modelo": "x1"}'
```

- Resultado esperado

```
HTTP/1.1 200 OK
X-Powered-By: tinyhttp
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, HEAD, PUT, PATCH, POST, DELETE
Access-Control-Allow-Headers: content-type
Content-Type: application/json
Date: Tue, 30 Jan 2024 02:16:28 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 51

{
  "id": "9",
  "marca": "bmw",
  "modelo": "x1"
```

- utilize 127.0.0.1, caso localhost ou 0.0.0.0 não funcione

Exercício 5

- Testar chamada do método HTTP PATCH via linha de comando

- LINUX/macOS

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X PATCH http://0.0.0.0:8080/carros/9 -d '{"id":"9","modelo":"x2"}'
```

- WINDOWS (CMD)

```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X PATCH http://localhost:8080/carros/9 -d '{"id":"9","marca":"bmw","modelo":"x2"}'
```

- Resultado esperado

```
HTTP/1.1 200 OK
X-Powered-By: tinyhttp
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, HEAD, PUT, PATCH, POST, DELETE
Access-Control-Allow-Headers: content-type
Content-Type: application/json
Date: Tue, 30 Jan 2024 02:25:15 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 51

{
  "id": "9",
  "marca": "bmw",
  "modelo": "x2"
}
```

- utilize 127.0.0.1, caso localhost ou 0.0.0.0 não funcione

Exercício 6

- Testar chamada do método HTTP DELETE via linha de comando

- LINUX/macOS

- ```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X DELETE http://0.0.0.0:8080/carros/9
```

- WINDOWS (CMD)

- ```
curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X DELETE http://localhost:8080/carros/9
```

- Resultado esperado

```
HTTP/1.1 200 OK
X-Powered-By: tinyhttp
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, HEAD, PUT, PATCH, POST, DELETE
Access-Control-Allow-Headers: content-type
Content-Type: application/json
Date: Tue, 30 Jan 2024 02:28:12 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 51

{
  "id": "9",
  "marca": "bmw",
  "modelo": "x2"
```

OBRIGADO!

<https://www.linkedin.com/in/josemariacesariojunior/>