Arquitetura REST e API's RESTful

Mentor: Carlos Júnior

1 - REST/RESTful

- O que é?
- Para que serve?

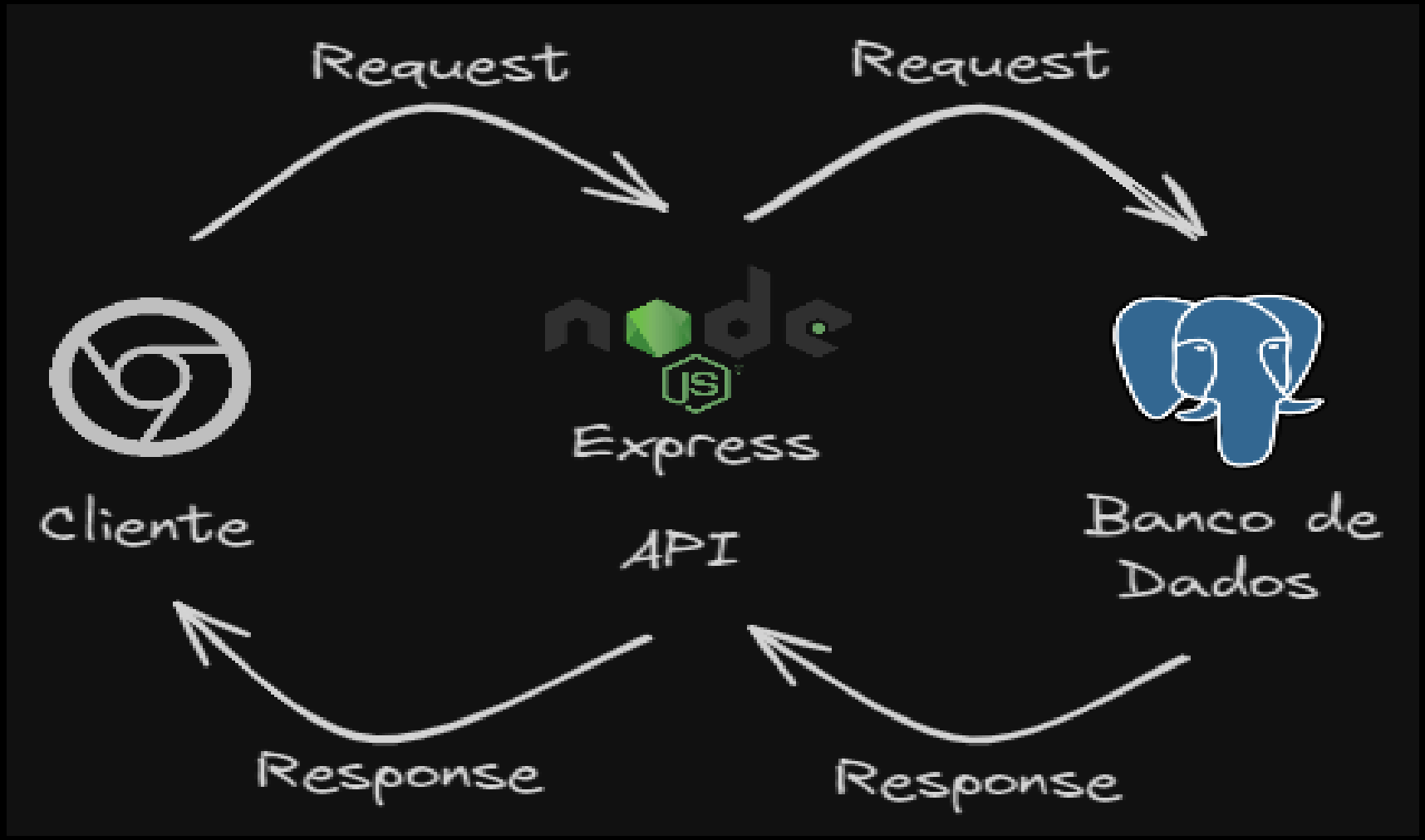
2 - API's RESTful

- O que são?
- Para que servem?

3 - Express.js

- Como criar uma API?

Relembrando...



REST/RESTful

Representational State Transfer (REST) é um conjunto de restrições/princípios arquiteturais para a criação de web services. Já RESTful é a implementação de todos esses princípios em uma aplicação.

Os princípios são:

- **Identificação de Recursos:** Cada recurso na API é identificado por um URI (Uniform Resource Identifier) único.
- **Utilização dos Métodos HTTP:** Os métodos HTTP padrão (GET, POST, PUT, DELETE) são usados para manipular recursos.
- **Representações dos Recursos:** Os recursos são representados em formatos padronizados e interoperáveis, como JSON, XML ou HTML.

REST/RESTful

- **Comunicação Stateless:** Cada solicitação à API deve conter todas as informações necessárias para ser processada, sem depender de estado armazenado no servidor.
- **HATEOAS (Hypermedia As The Engine Of Application State):** A API fornece links para recursos relacionados, permitindo que os clientes descubram a estrutura da API e naveguem pelos recursos de forma dinâmica.
- **Utilização Correta dos Códigos HTTP:** Os códigos HTTP padrão são usados para indicar o sucesso ou falha de uma solicitação.

REST/RESTful

Vantagens da arquitetura REST:

- **Interoperabilidade:** APIs RESTful podem ser facilmente integradas com outras APIs e sistemas.
- **Escalabilidade:** APIs RESTful podem ser facilmente dimensionadas para lidar com um grande número de solicitações.
- **Facilidade:** APIs RESTful são fáceis de serem compreendidas.

API's RESTful

Existem várias ferramentas que ajudam na construção de API's RESTful, segue abaixo algumas delas:

- **ASP.NET Core:** framework .NET (C#, F# e Visual Basic)
- **Spring Boot:** framework Java
- **Express:** framework JavaScript

Express.js

Usaremos o Express para criar uma API. Sendo assim, instale Node.js e o Express antes de prosseguir.

Criando uma API:

```
// app.js

const express = require('express')

const app = express()

// uma rota da api
app.get('/', (request, response) => {
    response.send('<h1> Olha meu response ai... </h1> ')
})

// definicao da porta que a api vai escutar
app.listen(3000, () => console.log('A API tá on na porta 3000!'))
```

Express.js

O parâmetro **request**, que é passado na callback de uma rota, é um objeto e possui diversos atributos. Os principais são:

- **req.headers:** Headers da requisição
- **req.body:** Corpo da requisição
- **req.params:** Parâmetros da rota
- **req.query:** Parâmetros da query string

Express.js

O parâmetro **response**, que é passado na callback de uma rota, é um objeto e possui diversos atributos que guardam funções. As principais são:

- **res.status():** Define o status da resposta
- **res.send():** Envia uma resposta simples
- **res.json():** Envia uma resposta JSON
- **res.end():** Encerra a resposta

Express.js

O Express também possui **middlewares**. Os **middlewares** são funções que tem acesso ao objeto de solicitação (request), o objeto de resposta (response), e a próxima função de **middleware** no ciclo solicitação-resposta da aplicação. Podendo efetuar as seguintes ações:

- Executar qualquer código
- Fazer mudanças nos objetos de solicitação e resposta
- Encerrar o ciclo de solicitação-resposta
- Chamar a próxima função de middleware na pilha

Express.js

Criando **middlewares** em rotas específicas:

```
// uma rota da api com um middleware
app.get('/', (request, response, next) => {
    console.log('Passando pelo middleware...')

    next()  // chama o proximo middleware
},
(request, response) => {
    response.send('<h1> Olha meu response ai... </h1> ')
})
```

Express.js

Criando **middlewares** globais:

```
// middleware global
app.use((request, response, next) => {
    console.log('Passando pelo middleware global...')

    next()  // chama o proximo middleware
})

// uma rota da api
app.get('/', (request, response) => {
    response.send('<h1> Olha meu response ai... </h1> ')
})
```

Express.js

Criando **middlewares** globais para uma rota específica:

```
// middleware global de todas as rotas derivadas da raíz (domain/)
app.use('/', (request, response, next) => {
    console.log('Passando pelo middleware global...')

    next()  // chama o proximo middleware
})

// uma rota da api
app.get('/', (request, response) => {
    response.send('<h1> Olha meu response ai... </h1> ')
})
```

Sua vez!

Lembra do mini projeto de JavaScript? Sua missão é refazê-lo, porém agora como uma API RESTful utilizando Node.js e Express.js. A organização dos arquivos fica a seu critério. Utilize persistência de dados em um arquivo no formato JSON. Sendo assim, siga os passos descritos a seguir:

1. Para listar as pessoas, sua API deve conter uma rota */pessoas*. Escolha qual o método da requisição e o status code das respostas, tendo em mente os motivos. A resposta deve ser um JSON no seguinte formato

   caso tenha pessoas:

```
[
    {
        "name": "Fulano",
        "email": "fulano@gmail.com",
        "phone": 40028922
    }
]
```

1. ○

   caso não tenha pessoas:

   ```
   []
   ```

2. Para adicionar uma pessoa, sua API deve conter uma rota */pessoas*. Escolha qual o método da requisição e o status code das respostas, tendo em mente os motivos. A pessoa a ser cadastrada deve ser enviada pelo corpo da requisição no formato JSON. Todos os campos da pessoa devem ser validados em termos de tipo. O email deve ser validado em termos de formato e regra de negócio (ele deve ser único). A resposta deve ser um JSON no seguinte formato

2. ○

caso tenha cadastrado com sucesso:

```
{
    "name": "Fulano",
    "email": "fulano@gmail.com",
    "phone": 40028922
}
```

caso tenha algum campo invalidado:

```
{
    "message": "mensagem especificando o motivo do erro"
}
```

Sua vez!

3. Para remover uma pessoa, sua API deve conter uma rota */pessoas*. Escolha qual o método da requisição e o status code das respostas, tendo em mente os motivos. O atributo que você escolheu para identificar a pessoa da operação deve ser enviado por parâmetro de caminho da requisição (path param). A resposta deve ser um JSON no seguinte formato

   caso tenha a pessoa solicitada:

```
{
    "name": "Fulano",
    "email": "fulano@gmail.com",
    "phone": 40028922
}
```

3. ○

   caso não tenha a pessoa solicitada:

```
{
    "message": "mensagem especificando o motivo do erro"
}
```

4. Para editar uma pessoa, sua API deve conter uma rota */pessoas*. Escolha qual o método da requisição e o status code das respostas, tendo em mente os motivos. O atributo que você escolheu para identificar a pessoa da operação deve ser enviado por parâmetro de caminho da requisição (path param). Todos os campos da pessoa devem ser informados no corpo da requisição no formato JSON, mesmo que ele não tenha modificações. A resposta deve ser um JSON no seguinte formato

4. ○

caso tenha a pessoa solicitada:

```
{
    "name": "Fulano",
    "email": "fulano@gmail.com",
    "phone": 40028922
}
```

caso não tenha a pessoa solicitada:

```
{
    "message": "mensagem especificando o motivo do erro"
}
```

Sua vez!

5. Para encontar uma pessoa, sua API deve conter uma rota */pessoas*. Escolha qual o método da requisição e o status code das respostas, tendo em mente os motivos. O atributo que você escolheu para identificar a pessoa da operação deve ser enviado por parâmetro de consulta da requisição (query param). A resposta deve ser um JSON no seguinte formato

   caso tenha a pessoa solicitada:

```
{
    "name": "Fulano",
    "email": "fulano@gmail.com",
    "phone": 40028922
}
```

5. ○

   caso não tenha a pessoa solicitada:

```
{
    "message": "mensagem especificando o motivo do erro"
}
```

Bônus

6. Para editar uma pessoa, sua API deve conter uma rota */pessoas*. Escolha qual o método da requisição e o status code das respostas, tendo em mente os motivos. O atributo que você escolheu para identificar a pessoa da operação deve ser enviado por parâmetro de caminho da requisição (path param). **Apenas** os campos da pessoa a serem modificados devem ser informados no corpo da requisição no formato JSON. A resposta deve ser um JSON no seguinte formato

6. ○

caso tenha a pessoa solicitada:

```
{
    "name": "Fulano",
    "email": "fulano@gmail.com",
    "phone": 40028922
}
```

caso não tenha a pessoa solicitada:

```
{
    "message": "mensagem especificando o motivo do erro"
}
```

Sua vez!

7. Altere a rota para encontrar uma pessoa criada na questão **5** para que ela receba, via query param, os três campos de uma pessoa: name, email e phone. A busca deve levar em consideração apenas os parâmetros passados, por exemplo: se for passado apenas os parâmetros name e phone, a busca deve levar em consideração apenas os dois. Se não for passado nenhum parâmetro, nenhuma pessoa deve ser retornada. A resposta deve ser um JSON no seguinte formato
   caso tenha pessoas solicitadas:

```
[
    {
        "name": "Fulano",
        "email": "fulano@gmail.com",
        "phone": 40028922
    }
]
```

7. ○

   caso não tenha pessoas solicitadas:

```
[]
```