

Introdução a API's e Express.js

Labenu_



O que vamos ver hoje? 🙄

- Revisão
- O que é uma API?
- Express.js
 - Instalação e Configuração
 - Sintaxe



Revisão para relembrar 🧠

Labenu_



Resumão

- Como já mencionamos, podemos caracterizar nosso **backend** como **um conjunto de funções acionadas por meio do protocolo HTTP**.
- Em módulos anteriores, criamos páginas que **enviam** requisições HTTP, utilizando o *Axios*
- Hoje, aprenderemos a construir programas que **recebam** essas requisições, utilizando o ***Express.js***

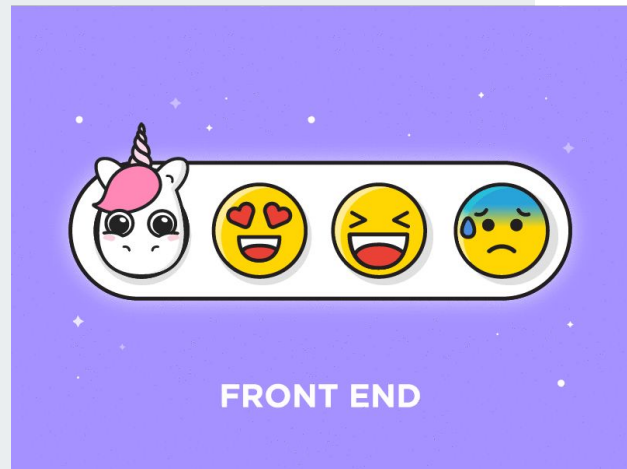


O que é uma API? 🙄

Labenu_



Introdução



- Application Programming Interface:
é a **área de contato** entre dois programas
(no nosso caso, o frontend e o backend)
- Normalmente, os programas se comunicam através do protocolo **HTTP** (HyperText Transfer Protocol)
 - Ex: O Frontend faz uma **requisição** (request) e o Backend retorna uma **resposta** (response)
- Exemplo: [Labefy](#)



Express.js



Labenu_



Instalação e Configuração

Labenu_



Instalação

- Assim como o *Axios* e o próprio *React*, o *Express* é uma biblioteca de código JS, adotada por simplificar o uso de seus recursos nativos. Logo, para acessá-la, devemos criar um pacote do **Node**
- Para adicionar o *Express* como dependência, usamos o comando:
`npm install express`
- Como nossos programas serão escritos em Typescript, devemos também adicionar a versão tipada dessa biblioteca como dependência de desenvolvimento:

```
npm install @types/express --save-dev
```



Instalação

- Instalaremos, também, uma *lib* auxiliar, o ***cors***. Ela nos permitirá enviar requisições de uma página estática (front), hospedada localmente, para um servidor HTTP (back), também executado localmente

```
npm install cors  
npm install @types/cors --save-dev
```



Configuração

- O próximo passo é inicializar esse servidor. Fazemos isso importando e invocando, no nosso index.ts, a função que a *lib* exporta por padrão

```
import express from 'express'

const app = express()
```



Configuração

- Em seguida, configuramos dois serviços (ou *middlewares*, mas por enquanto podemos chamar de **linhas mágicas**): um para converter o body das nossas respostas para o formato json, e outro para evitar o erro de CORS (antes, importaremos a *lib* que tem esse mesmo nome)

```
import express from 'express'
import cors from 'cors'

const app = express()

app.use(express.json())
app.use(cors())
```

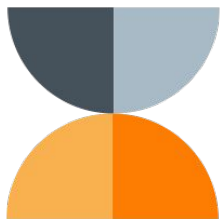


Configuração

- Finalmente, faremos o servidor escutar alguma porta da nossa máquina. O método que usamos para isso recebe, como segundo parâmetro, uma *callback* que podemos utilizar para sinalizar que a aplicação está pronta

```
app.listen(3003, () => {  
  console.log("Server is running in http://localhost:3003");  
});
```





PAUSA PARA RELAXAR



Sintaxe

Labenu_



Construindo Endpoints

- Com o express instalado e configurado, podemos criar nosso primeiro endpoint, seguindo o padrão:

```
app.method(path, handler)
```

- No modelo acima, *handler* é a função que estamos acionando ao bater no endpoint. Ela recebe dois parâmetros representando, respectivamente, a requisição recebida e a resposta enviada

```
app.get('/', (req: Request, res: Response) => {  
  res.send('Hello, world!')  
})
```



Request e Response

Labenu_



Request e Response

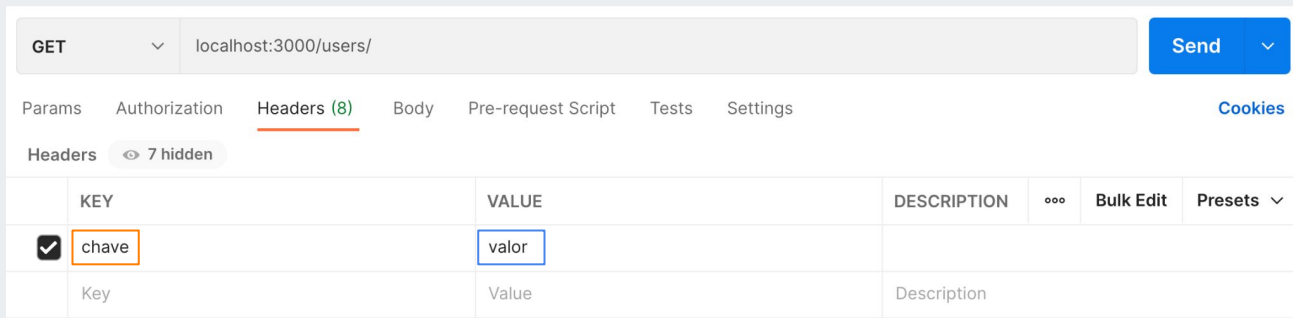
As principais propriedades do parâmetro **req** representam as diferentes formas pelas quais o front pode enviar dados:

- **req.headers** - Parâmetros de cabeçalho
- **req.body** - Parâmetros de corpo
- **req.query** - Parâmetros de consulta
- **req.params** - Parâmetros de caminho



Request e Response

- **req.headers** - acessa chaves passadas no cabeçalho da requisição



GET localhost:3000/users/ Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Headers 7 hidden

	KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	chave	valor				
	Key	Value	Description			

```
app.get('/users', (req: Request, res: Response) => {  
  if (req.headers.chave) === "valor" {  
    res.send("A chave vale: valor")  
  } else {  
    res.send("A chave não vale valor")  
  }  
})
```



Request e Response

- **req.body** - acessa chaves passadas no corpo da requisição

```
{  
  "chave": "valor"  
}
```

```
app.get('/users', (req: Request, res: Response) => {  
  if (req.body.chave) === "valor") {  
    res.send("A chave vale: valor")  
  } else {  
    res.send("A chave não vale valor")  
  }  
})
```



Request e Response

- **req.query** - acessa chaves passadas por *query parameters* (ex: <https://endereco-da-api.com/users?chave=valor>)

```
app.get('/users, (req: Request, res: Response) => {  
  if (req.query.chave) === "valor" {  
    res.send("A chave vale: valor")  
  } else {  
    res.send("A chave não vale valor")  
  }  
})
```



Request e Response

- **req.params** - acessa valores passadas por *path parameters*. Diferente dos parâmetros de caminho, o valor fica **direto no URL** e a chave é descrita **no endpoint** (ex: **https://endereco-da-api.com/users/1**)

```
app.get('/users/:age', (req: Request, res: Response) => {  
  if (Number(req.params.age) % 2 === 0) {  
    res.send("Sua idade é par")  
  } else {  
    res.send("Sua idade é ímpar")  
  }  
})
```



Request e Response

De maneira análoga, os principais métodos do parâmetro **res** são:

- **res.status(n)** - envia uma resposta com status **n**
- **res.send(x)** - responde a requisição com o objeto **x**
- **res.end()** - encerra a requisição sem um *body* na resposta



Exercício 1

A partir do template fornecido, vamos reproduzir a API do Labefy!

Começaremos, na aula de hoje, por quatro endpoints:

- Get All Playlists
- Get Playlist Tracks
- Delete Playlist
- Delete Track

Resumo

Labenu_



Resumo

- **Express** é uma lib amplamente utilizada para criação de API's
- A criação de endpoints segue o padrão:
`app.method(path, handler)`
- Os *handlers* recebem dois parâmetros: o primeiro representando a requisição e o segundo, a resposta



Dúvidas?





Obrigado!