

Serviços no Backend

Labenu_



O que vamos ver hoje? 🙄

- Serviços externos através do axios;
- Nodemailer
- Deploy do Backend usando Heroku.
- Criando documentações com o postman



Integrando APIs no Backend

Labenu_



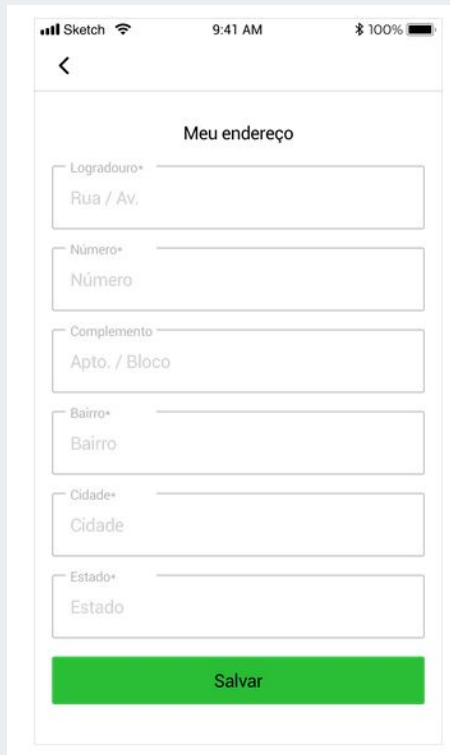
Integrando APIs no Backend

- Integração de APIs não é exatamente uma novidade, pelo contrário, é algo que vocês já sabem fazer desde o React
- A novidade de hoje é algo que talvez vocês não tenham considerado ainda: **podemos integrar nossa própria API com APIs de terceiros**
- E como fazer isso? Da mesma forma que fizemos no front: utilizando o **Axios**



Integrando APIs no Backend

- Para exemplificar, vamos relembrar o projeto final do front: o Labe Foods
- Nele, o cadastro exigia uma série de campos que compunham o endereço do usuário



A mobile app mockup for address registration. The screen shows a form titled "Meu endereço" with the following fields: "Logradouro*" (containing "Rua / Av."), "Número*" (containing "Número"), "Complemento" (containing "Apto. / Bloco"), "Bairro*" (containing "Bairro"), "Cidade*" (containing "Cidade"), and "Estado*" (containing "Estado"). A green "Salvar" button is at the bottom.



Integrando APIs no Backend

- Nesse formulário, as informações de *logradouro*, *bairro*, *cidade* e *estado* podem ser obtidas a partir do CEP da residência, através do webservice **ViaCep**

A screenshot of a web browser window. The address bar shows the URL 'https://viacep.com.br/ws/05424150/json/'. The main content area displays a JSON object with the following fields: 'cep', 'logradouro', 'complemento', 'bairro', 'localidade', 'uf', 'ibge', 'gia', 'ddd', and 'siafi'.

```
{
  "cep": "05424-150",
  "logradouro": "Rua Pais Leme",
  "complemento": "lado ímpar",
  "bairro": "Pinheiros",
  "localidade": "São Paulo",
  "uf": "SP",
  "ibge": "3550308",
  "gia": "1004",
  "ddd": "11",
  "siafi": "7107"
}
```

<https://viacep.com.br/ws/05424150/json/>



Integrando API's no Backend

- Com isso em mente, podemos construir um **middleware** que receba apenas o CEP, número e complemento da residência e, após consultar as demais informações, faça uma requisição para o endpoint de adicionar endereço

PUT Add Adress

`https://us-central1-missao-newton.cloudfunctions.net/{{appName}}/address`

Este endpoint é autenticado. Deve receber um token de usuário no parâmetro `auth` do header.

Este endpoint realiza o cadastro **ou a edição** do endereço de um usuário. Todos os usuários devem cadastrar um endereço, ou os outros endpoints não funcionarão.

Todos os parâmetros são obrigatórios, com exceção do `complement`.

Em caso de sucesso, ele retorna as informações do usuário e um novo token de acesso, que deve ser armazenado e enviado em outras requisições pelo header `auth`.

ATENÇÃO: MESMO QUE VOCÊ JÁ POSSUA UM TOKEN SALVO, VOCÊ DEVE SUBSTITUI-LO PELO NOVO TOKEN RETORNADO POR ESSE ENDPOINT

HEADERS

auth	<code>{{token}}</code>
Content-Type	<code>application/json</code>

BODY raw

```
{
  "street": "R. Afonso Braz",
  "number": "177",
  "neighbourhood": "Vila N. Conceição",
  "city": "São Paulo",
  "state": "SP",
  "complement": "71"
}
```





Exercício 1

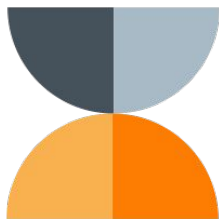
Crie uma pasta *services*, com uma função **getAddressInfo** que cumpra os seguintes requisitos:

- Receber como entrada um CEP (apenas dígitos numéricos)
- Fazer uma requisição para <https://viacep.com.br/ws/:cep/json/>
- Retornar o logradouro correspondente, juntamente com bairro, cidade e estado



Pausa para relaxar 🤪

5 min



- Nossas APIs, enquanto interfaces de acesso para dados, podem também consultar outras APIs como fonte;
- Vimos como fazer isso utilizando a integração com o ViaCep.



Nodemailer - Enviando dados para fora da API

Labenu_



Enviando o e-mail

- Uma outra forma de interagir com itens externos é utilizando protocolos diferentes do HTTP em nossa aplicação. Em nosso caso, utilizaremos o **SMTP**.
- O SMTP é o protocolo utilizado para envios de e-mail e, felizmente, temos uma lib do JS/TS perfeita para fazer isso do lado do servidor: o **nodemailer**.



Enviando o e-mail

- O nodemailer é uma lib do node que podemos utilizar para enviar e receber e-mails fora da interface dos clientes de e-mail conhecidos.
- Instalá-lo é bem simples, é só utilizar os comandos abaixo:

```
npm install nodemailer  
npm install @types/nodemailer -D
```



Configurando o nodemailer

- Para conseguirmos enviar uma mensagem de correio eletrônico pelo node, precisamos de **duas** coisas
 - Configurar o servidor de e-mail responsável pelo envio da mensagem;
 - Preparar o conteúdo da mensagem para envio.
- Vamos ver como essas duas coisas acontecem:



Configurando o nodemailer

- Criaremos um **transporter**, que é o objeto utilizado para enviar uma mensagem. Ele recebe as configurações de acesso e segurança necessárias para o envio. As informações mais importantes são:

```
import nodemailer from "nodemailer";
import dotenv from "dotenv"

dotenv.config()

const transporter = nodemailer.createTransport({
  host: "smtp.gmail.com",
  port: 587,
  secure: false,
  auth: {
    user: process.env.NODEMAILER_USER,
    pass: process.env.NODEMAILER_PASS
  },
  tls: { ciphers: "SSLv3" }
})

export default transporter
```



Configurando o nodemailer

- **host:** Determina o endereço do servidor SMTP utilizado;
- **port:** Determina a porta do servidor que utilizaremos para acessá-lo;
- **secure:** Diz se utilizaremos ou não SSL;
- **auth:** Dados de acesso ao server de e-mail;
- **tls:** Configurações opcionais

```
import nodemailer from "nodemailer";
import dotenv from "dotenv"

dotenv.config()

const transporter = nodemailer.createTransport({
  host: "smtp.gmail.com",
  port: 587,
  secure: false,
  auth: {
    user: process.env.NODEMAILER_USER,
    pass: process.env.NODEMAILER_PASS
  },
  tls: { ciphers: "SSLv3" }
})

export default transporter
```



Escrevendo um e-mail

```
const info = await transporter.sendMail({  
  from: "<*****@email.com>",  
  to: "exemplo@email.com",  
  subject: "Mensagem de exemplo",  
  text: "Este é um texto de exemplo",  
  html: "<p>Exemplo em HTML</p>"  
})
```


Assim como as configurações de acesso, o e-mail também é feito utilizando um objeto, que define as propriedades daquela mensagem específica. A seguir, vemos o que cada uma faz.



Escrevendo um e-mail

```
const info = await transporter.sendMail({
  from: "<*****@email.com>",
  to: "exemplo@email.com",
  subject: "Mensagem de exemplo",
  text: "Este é um texto de exemplo",
  html: "<p>Exemplo em HTML</p>"
})
```

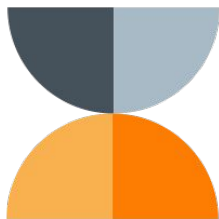
- **from:** Remetente;
- **to:** Destinatário;
- **subject:** Campo de assunto do e-mail;
- **text:** O texto que aparece na versão minificada do e-mail;
- **html:** O corpo do e-mail.

Vamos ver na prática! 



Pausa para relaxar 🧘

5 min



- Até aqui nossa aplicação estava limitada a interagir apenas com suas fontes de dados. Vimos que também é possível utilizar dados externos no **backend**;
- Começamos a construir um fluxo de **envio de e-mails**.



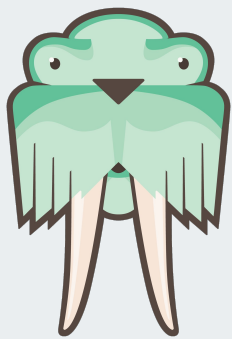
Deploy no Heroku

Labenu_



Heroku

- O front possui uma ferramenta de prototipagem rápida para páginas estáticas, o **Surge**.
- O back possui uma ferramenta análoga para publicação de servidores: o **Heroku**



```
npm i -g surge
```



```
npm i -g heroku
```

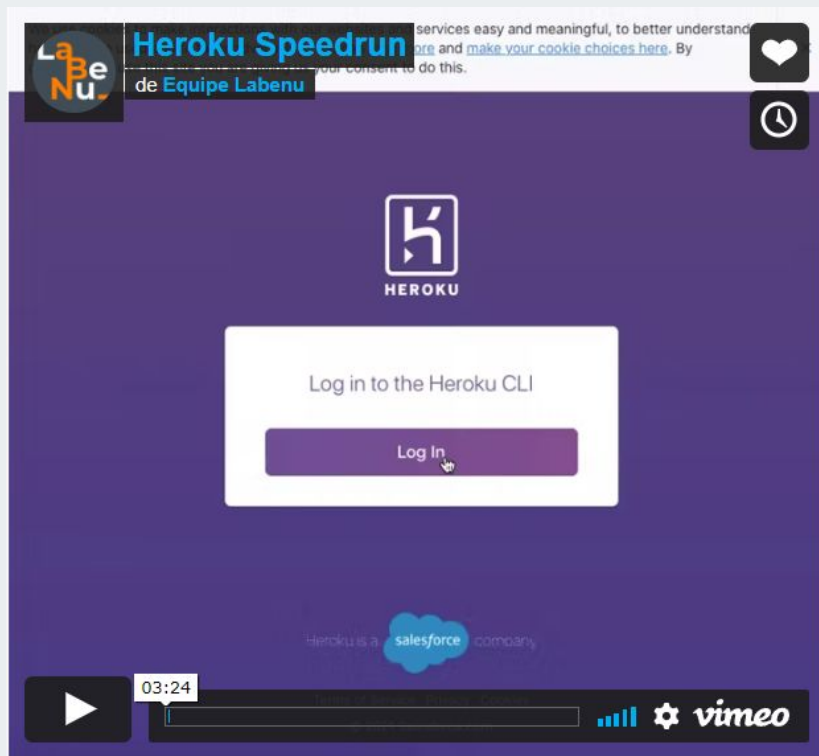


Heroku

Etapas para deploy no Heroku:

- Criar conta no [site](#)
- Criar novo app
- *Settings* (env, buildpack)
- `npm i -g heroku`
- `heroku login`
- `git init` (ou `git clone`)
- `heroku git:remote app-name`
- `git add .`
- `git commit -am 'deploy'`
- `git push heroku master`
- `heroku logs --tail`

[Video tutorial](#)



Heroku

Problemas comuns:

- Bug em devDependencies:
 - listar todas as lib's como dependencies
- Falha ao escutar a porta 3003
 - passar a porta como variável de ambiente. Esta variável precisa, obrigatoriamente se chamar **PORT**

```
...  
  
const { PORT = 3003 } = process.env  
  
app.listen(PORT, () => {  
  console.log(`Server is running on port ${PORT}`);  
});
```



Documentação no Postman

Labenu_



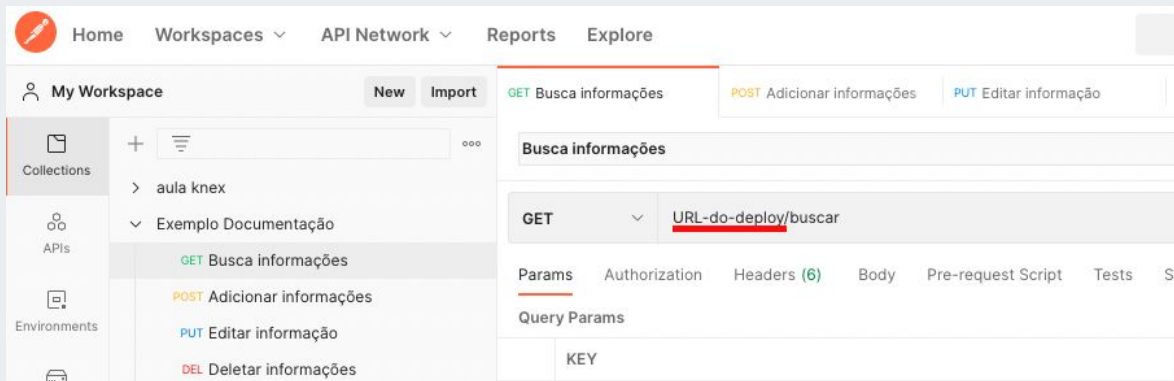
Postman

- A documentação é o “guia” do seu backend, sem ela é necessário “descobrir” como ele funciona. O que acaba consumindo tempo
- Para evitar isso, criamos as documentações de forma **mais clara e sucinta** que conseguirmos
- O próprio **postman** oferece essa ferramenta



Postman

- Coloque a URL fornecida no deploy em seus endpoints:



- Nas opções da coleção e visualize a documentação



Postman

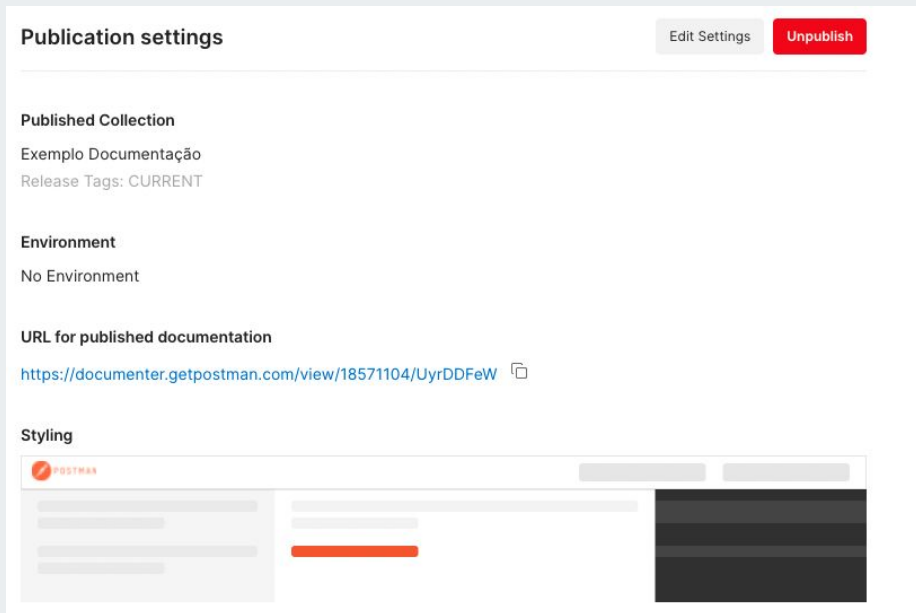
- Na documentação, clique em publicar. O site do postman será aberto e você poderá editar algumas informações antes de publicar


The screenshot shows the Postman API documentation interface for a collection named "Exemplo Documentação". At the top, there is a header bar with the collection name on the left and three icons on the right: a fork icon with "0", a "View Collection" button, and a "Publish" button with a red underline. Below the header, there is a sub-header with "Release Tag CURRENT", "Language cURL", and a settings gear icon. The main content area is divided into two columns. The left column has the title "Exemplo Documentação" followed by a description "Make things easier for your teammates with a complete collection description.". Below this, there is a section for a "GET Busca informações" endpoint. It includes a text input field containing "URL-do-deploy/buscar" and a blue "Open Request" button with a right arrow. Below the input field is another description: "Make things easier for your teammates with a complete request description.". The right column is titled "JUMP TO" and lists four items: "Introduction", "GET Busca informações", "POST Adicionar informações", "PUT Editar informação", and "DEL Deletar informações". Each item is preceded by a small colored square corresponding to its HTTP method.



Postman

- O postman fornece um link para você divulgar a documentação; insira esse link no readme dos seus projetos!



Vamos ver na prática! 



Dúvidas? 🧐

Labenu_



Resumo

Labenu_



Resumo



- Vimos que é possível utilizar dados externos no **backend**;
- Começamos a construir um fluxo de **cadastro de endereços**, utilizando a integração com o ViaCep.



Resumo

- Conhecemos o **nodemailer**, biblioteca que nos permite utilizar o protocolo SMTP para enviar e-mails;
- Aprendemos a configurá-lo para envio.
- Vimos que nossa API pode ser uma intermediária entre APIs já existentes e uma aplicação frontend. Chamamos esse tipo de intermédio de **middleware**





Obrigado!