Fundamentos de React

Professor: Wesley Bruno Barbosa Silva

email: [wesley.silva@prof.infnet.edu.br](mailto:wesley.silva@prof.infnet.edu.br)

# Aula - Terça - 23/07/24 - Julho

### Competências:

* Utilizar Javascript para desenvolver aplicações ReactJS;
* Desenvolver componentes em uma interface;
* Apresentar dados dinâmicos em uma interface ReactJS;
* Implementar condicionais e lista em uma aplicação ReactJS.

### ETAPA 1

Ambiente **REPLIT – Node.js**

Variáveis - Array – Funções - Tudo igual ao java script.

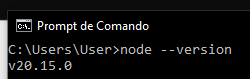
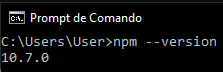
# Aula - Quinta - 25/07/24 – Julho

### RESUMO DA AULA:

**CRIAR UM PROJETO NO REACT E SUBIR PARA O GIT HUB.**

Versão do **NOTE.JS**. No terminal digite:

node –version npm --version

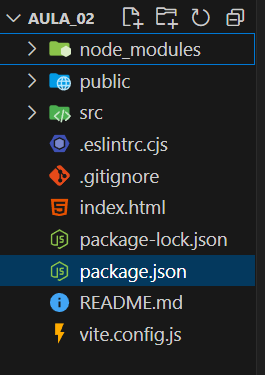
npm install –global yarn



## Criar projeto

No terminal, dentro da pasta escolhida ou no vs code, no terminal, digite:

npm create vite@latest

Após criado o projeto então o abrimos no VS CODE e dentro dele no terminal execute os comandos abaixo:  
npm install (são os pacotes no react)

npm run dev (inicializa a aplicação)

Estrutura final do projeto

# Aula - Terça - 30/07/24 – Agosto

### REACT introdução

### Fast Food – Landing page

<https://www.figma.com/design/PJruXxCaSO8uC2nhr87WXg/Infnet_01-Fast-Food?m=dev>

# Aula - Quinta - 01/08/24 – Agosto

### Fast Food – Landing page

<https://www.figma.com/design/PJruXxCaSO8uC2nhr87WXg/Infnet_01-Fast-Food?m=dev>

### LEITURA EM AULA TP1

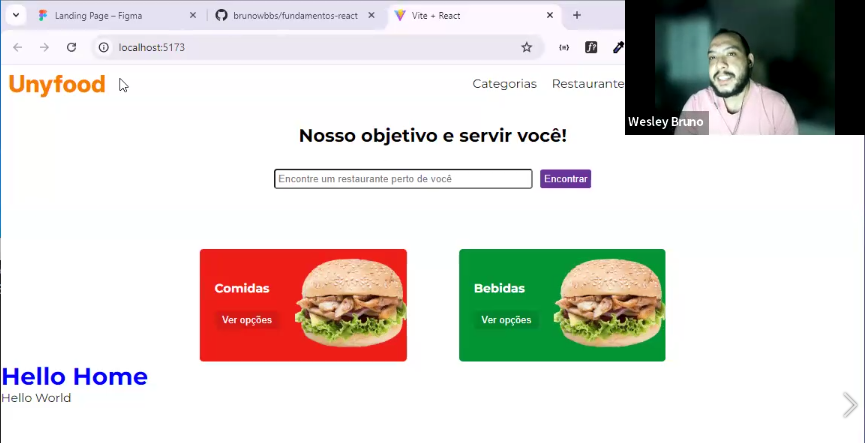
# Aula - Terça - 06/08/24 – Agosto

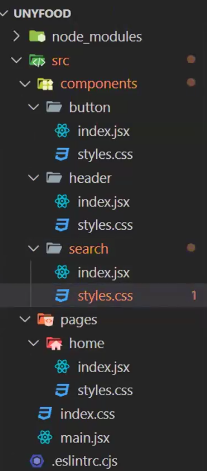
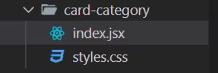
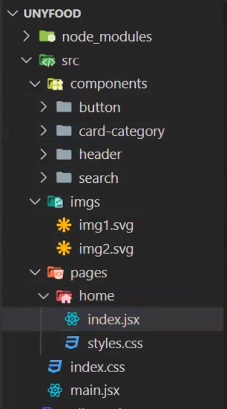
### Unyfood – Landing page

<https://www.figma.com/design/PpT3B7XSgQuKqC5ha97xPK/Infnet_02-Landing-Page?node-id=0-1&m=dev>

Styles com parâmetros (0:32 min de aula) passados para o componente

Componente com styles dinâmicos



# Aula - Quinta - 08/08/24 – Agosto

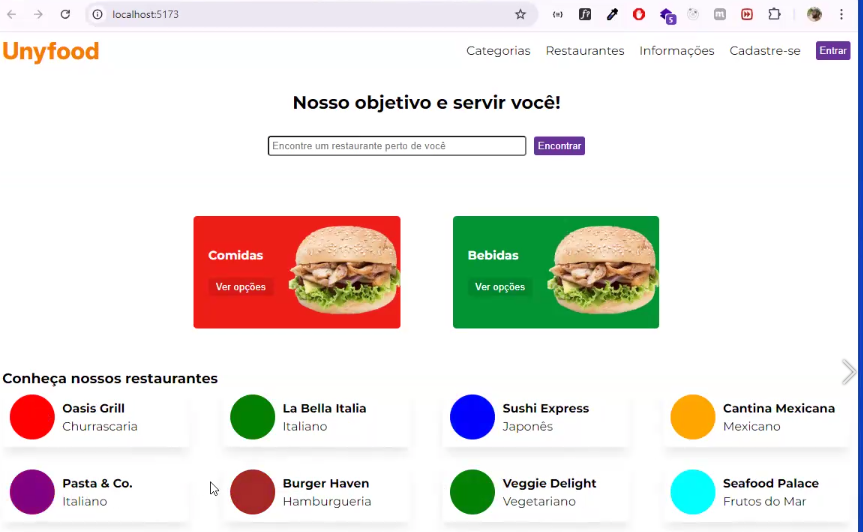
### Unyfood – Landing page

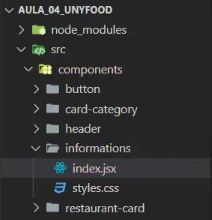
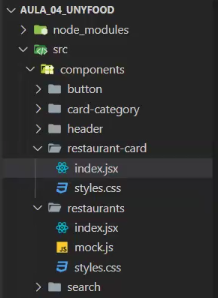
<https://www.figma.com/design/PpT3B7XSgQuKqC5ha97xPK/Infnet_02-Landing-Page?node-id=0-1&m=dev>



### (0:34 min) RestaurantCard

### (0:43 min) Uso da função map para iterar sobre uma lista de card a ser populado





# Aula - Terça - 13/08/24 - Agosto

### States - Hook

Exemplo de um contador atualizando o valor quando o usuário pressiona um botão de forma dinâmica.

(01h 05 min) Exemplo de cores aleatórias usando hexadecimal.

# Aula - Quinta - 15/08/24 – Agosto

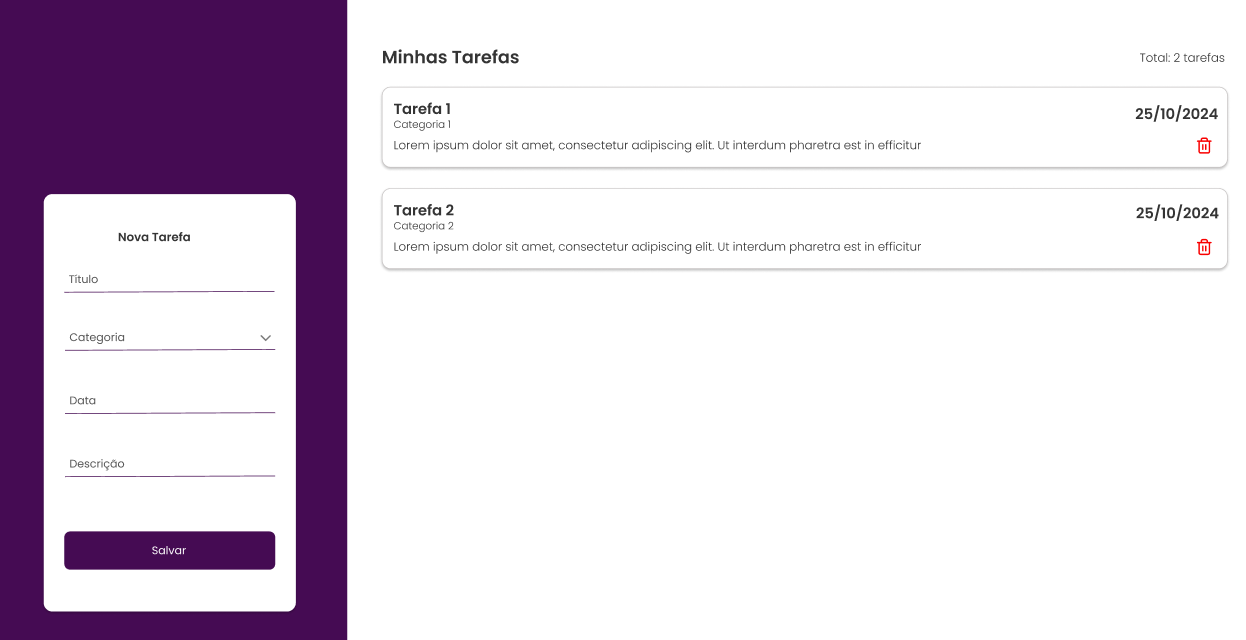
### States – Calculadora

### ENTREGA TP1

# Aula - Terça - 20/08/24 – Agosto

### To do List

<https://www.figma.com/design/mqckVkN5kzB9CLRaZhW2ph/ToDo-App?node-id=0-1&t=WMkfA8tDteU1fqSZ-0>



# Aula - Quinta - 22/08/24 – Agosto

### Continuação To do List

### Navegador > Inspecionar > Local storage

* Armazena dados no navegador semelhante a um banco de dados
* Funciona ainda se o navegador for fechado
* Limite de 5mb
* Utilizar somente para teste

### Use Effects (chave/valor)

* O valor é o primeiro parâmetro (**recuperarTarefas()**);
* A chave é o segundo parâmetro (**[]**), ou seja, é a condição para a função ser executada;

Serve para tratar os efeitos que causam erros.

Podemos ter inúmeros **useEffects** com funções e condições diferentes.

Importação do useEffects:

import { useState, useEffect } from 'react';

No exemplo abaixo vemos como carregar os dados da página uma única vez para que não fiquem sendo executados de forma recursiva.

// Quando temos um array vazio e a função está dentro do carregamento do componente a mesma será executada uma só vez. A função recuperarTarefas() é executada somente uma vez.

useEffect(()=>{

      recuperarTarefas();

}, []);

Acionar useEffect quando o campo [titulo] for alterado.

// Toda vez que o parâmetro título for alterado a função recuperarTarefas será executada

useEffect(()=>{

  recuperarTarefas();

}, [titulo]);

Adicionar useEffect quando a função for declarada

// Quando colocamos uma função “apagarTarefa” ela é executada quando a função for declarada

useEffect(()=>{

  if (titulo === "abc") {

    recuperarTarefas();

  }

}, [titulo, categoria, apagarTarefa]);

Executa toda vez que o usuário fizer alguma ação.

useEffect(()=>{

  recuperarTarefas();

});

Ação ao arrastar um documento para a tela

useEffect(()=>{

  abrirOpcoes();

}, [novoArquivo]);

Executar **useEffects** em todas menos na primeira renderização

// Executa todas as vezes que o campo titulo for alterado, exceto na primeira renderização

export function Home(){

   const [primeiraRenderizacao, setPrimeiraRenderizacao] = useState(true);

   useEffect(()=>{

      if (primeiraRenderizacao) {

         setPrimeiraRenderizacao(false);

      }else{

         recuperarTarefas();

      }

   }, [titulo]);

### Bilbioteca NPM

<https://www.npmjs.com/>

olhar a quantidade de download semanal

quando forma as últimas publicações

olhar as correções dos erros (isues)

* React router DOM
* react-iconS

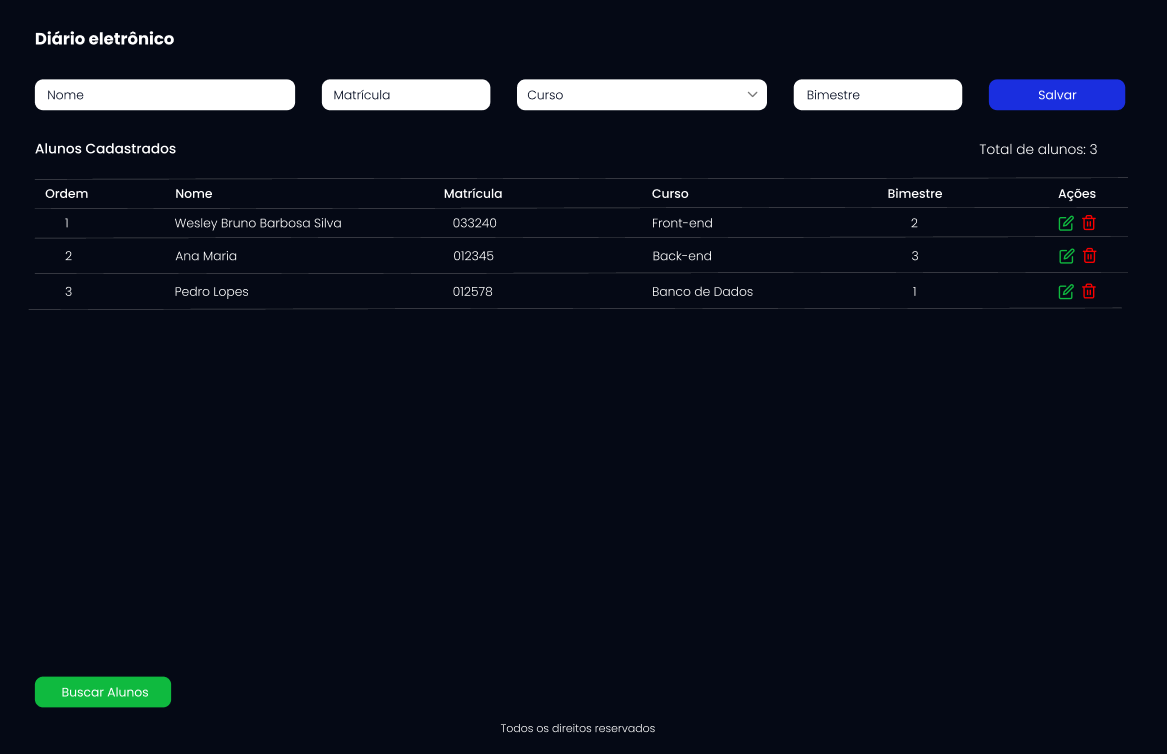
### Biblioteca de componentes

<https://github.com/brunowbbs/ui-components.git>

# Aula - Terça - 27/08/24 – Agosto

### Diário Eletrônico

<https://www.figma.com/design/6OAG9XRbimb5HMMWFNi4sl/Diario-eletronico?node-id=1-10&node-type=FRAME&t=WAQgcyPAQU9vgc0O-0>



**PASSOS**

* Olhar configurações externas (fontes e imagens)
* Resetar o documento (margin, padding e box-sizing)
* Criar variáveis de cores

**Tabelas**

<https://www.w3schools.com/html/html_tables.asp>

# Aula - Quinta - 29/08/24 – Agosto

### ENTREGA TP2 (02/09/24)

## Requisições HTTP

### Endereço + tipo de operação = data + status

* Comunicação do Front End com Back End
* Geralmente é via API
* Cliente via API solicita ao SERVIDOR o acesso aos dados, o SERVIDOR le esses dados e retorna pro cliente os dados obtidos.
* O cliente faz essa solicitação acessando o endereço (ex: <https://lojadigital.com/produtos>)
* O banco de dados pode fazer quatro operações básicas CRUD (CREATE, READ, UPDATE, DELETE)
* Ao fazer a operação devo informar qual operação vai ser feita:
* Cliente faz o POST (CREATE) para o SERVIDOR, que fará as alterações no banco de dados
* Cliente faz o GET (READ) para o SERVIDOR, que fará as alterações no banco de dados
* Cliente faz o UPDATE (PUT/PATCH) para o SERVIDOR, que fará as alterações no banco de dados
* Cliente faz o DELETE (DELETE) para o SERVIDOR, que fará as alterações no banco de dados

### Requisição = REQUEST

### Retorno = RESPONSE

* Além da resposta da requisição o SERVIDOR retorna um STATUS (100 até 599)

Site com todos os STATUS

<https://http.cat/>

400 ao 499 erros no back end (a requisição do front tem algum problema)

* O retorno do SERVIDOR é um data do tipo JSON/XML geralmente, para API que retorna para o CLIENTE
* Converter imagem

<https://www.base64-image.de/>

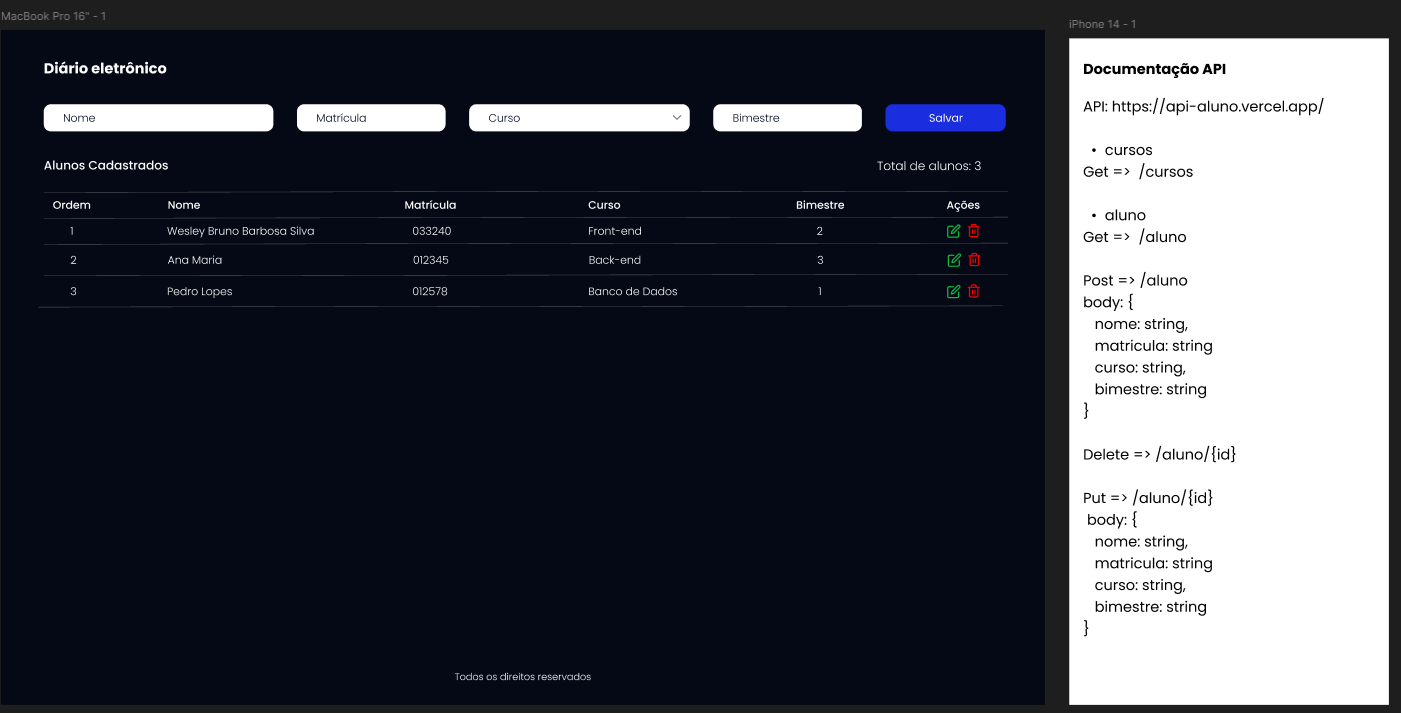
### Salvar os dados no banco de dados via API

API: <https://api-aluno.vercel.app/>

ROTA: caminho que o BACK END desenvolveu

Acessar cursos API/cursos

<https://www.figma.com/design/HutAD3itDroy7Iif6MeiBB/Di%C3%A1rio-eletr%C3%B4nico?node-id=902-481&node-type=FRAME&t=2R3JCdpB9tw5xRuR-0>



## API – Diário Eletrônico

URL = <https://api-aluno.vercel.app/>

#### Rotas

**Get** => /cursos

**Get** => /aluno

**Post** => /aluno

body: {

nome: string,

matricula: string

curso: string,

bimestre: string

}

**Delete** => /aluno/{id}

**Put** => /aluno/{id}

body: {

nome: string,

matricula: string

curso: string,

bimestre: string

}

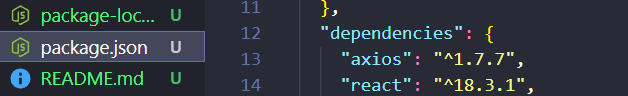
### Implementar via java script – FETCH

### Implementar via axios

<https://www.npmjs.com/package/axios>

**Instalar Biblioteca**

npm i axios

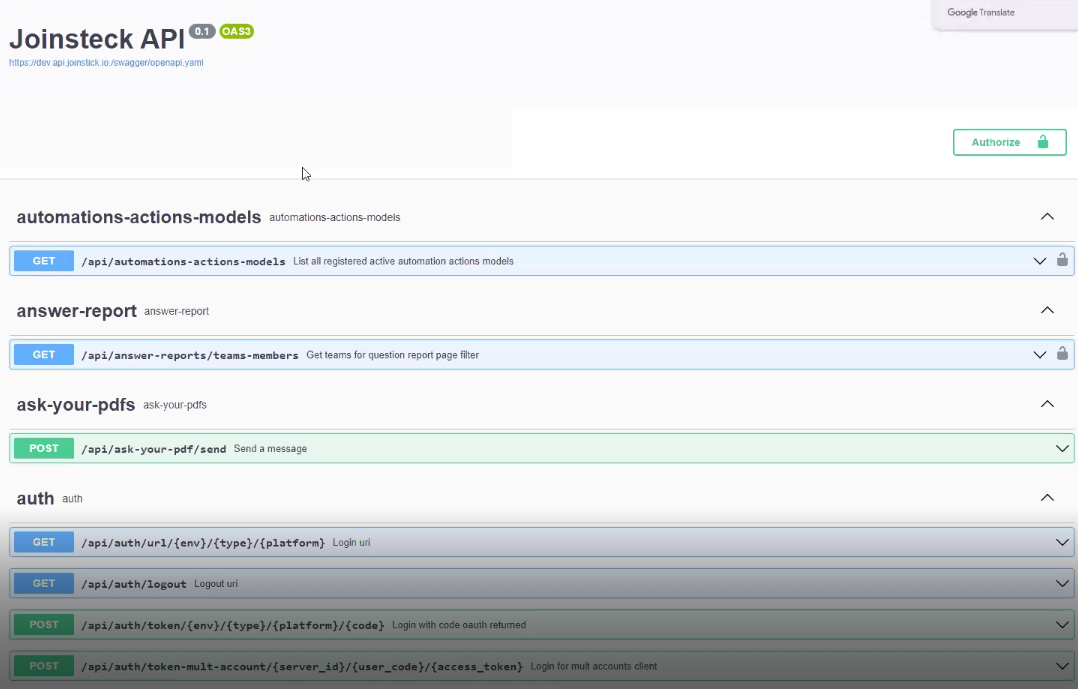


**Utilização**

import axios from 'axios';

**Exemplo documentação API Sweeger**

<https://swagger.io/>



A função **buscarAlunos** pode ser implementada em um botão no evento **onClick**, caso não queiramos que ela carregue inicialmente e sim somente com uma ação do usuário.

Caso não use **useEffects** causa loop infinito.

Primeiro o componente é montado e depois disso o **useEffect** é executado.

function buscarAlunos() {

  const response = axios.get("https://api-aluno.vercel.app/aluno");

}

// Assim que o componente for montado eu quero fazer a requisição, executa só uma vez

useEffect(() => {

  buscarAlunos();

}, []);

Toda vez que a const indexSelecionado fora alterada o **useEffect** é executado.

useEffect(() => {

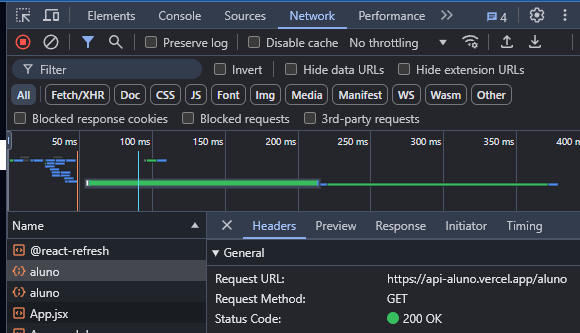
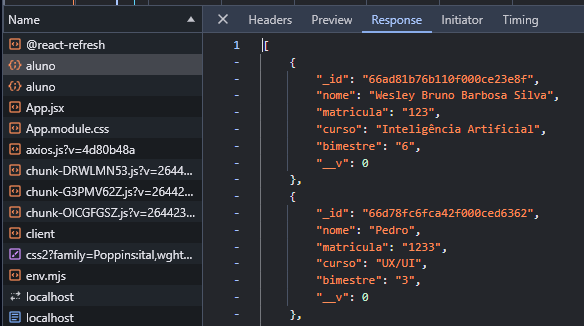
    buscarAlunos();

  }, [indexSelecionado]);

### UseEfects

Controla a quantidade de renderizações

### Request- resultado

## GET (buscarAluno) - Promisses

Funções que precisam de um tempo para ser executada. Desse modo o java script espera a função ser executada de modo assíncrona e não pula para a próxima linha.

Se o retorno da função é uma **promisse**, deve ser adicionado **ASYNC / AWAIT**.

Sempre que houver uma **promisse** devemos usar o **try**, para não exibir a tela branca para o usuário

async function buscarAlunos() {

  try {

    const response = await axios.get("https://api-aluno.vercel.app/aluno");

    setAlunos(response.data);

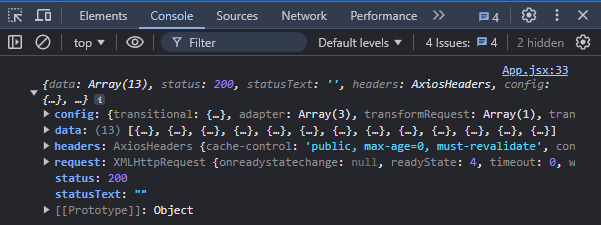
  } catch (error) {

    alert("Weeo ao buscar dados dos alunos")

  }

}

**Console (Data tem o RESPONSE(dados) + Status)**



## POST (addAluno) - Promisses

Adiciona um novo aluno e depois atualiza a lista de alunos

async function addAluno(*event*) {

*event*.preventDefault();

  try {

    await axios.post("https://api-aluno.vercel.app/aluno", {

      nome: formData.nome,

      matricula: formData.matricula,

      curso: formData.curso,

      bimestre: formData.bimestre,

    });

    buscarAlunos();

    alert("Sucesso ao salvar aluno");

  } catch (error) {

    alert("Erro ao salvar aluno");

  }

}

### <https://socket.io/>

### API = <https://github.com/brunowbbs/api-aluno>

### API FAKE(teste) = <https://dummyjson.com/docs>

Ideal para praticar

# Aula - Terça - 03/09/24 – Setembro

### PUT (EDITAR)

async function editarAluno(*event*) {

*event*.preventDefault();

  try {

    await axios.put(`https://api-aluno.vercel.app/aluno/${idSelecionado}`, formData

      // ou

      // {

      // nome:formData.nome,

      // matricula:formData.matricula,

      // curso:formData.curso,

      // bimestre:formData.bimestre,

      // }

    );

    setFormData({

      nome: "",

      matricula: "",

      curso: "",

      bimestre: "",

    })

    setIdSelecionado("");

    alert("Aluno editado com sucesso!");

    buscarAlunos();

  } catch (error) {

    alert("Erro ao editar dados do aluno")

  }

}

### DELETE (APAGAR)

async function removerAluno(*id*) {

  try {

    await axios.delete(`https://api-aluno.vercel.app/aluno/${*id*}`);

    buscarAlunos();

    alert("Sucesso ao remover aluno");

  } catch (error) {

    alert("Erro ao remover aluno")

  }

}

### MODAL

* **CSS**

Podemos adicionar um **className** na chamada do modal pra uma estilização particular.

<*Modal* *isOpen*={true} *className*={customStyles}>

    <h2>Confirmação</h2>

    <p>Deseja remover o item selecionado</p>

    <button>Sim</button>

    <button>Não</button>

</*Modal*>

* **Link**

<https://www.npmjs.com/package/react-modal?activeTab=readme>

* **Instalar**

npm i react-modal

* **Documentação**

<https://github.com/reactjs/react-modal?tab=readme-ov-file#api-documentation>

* **Import e Set**

import Modal from './react-modal'

// CSS Default

const customStyles = {

  content: {

    top: '50%',

    left: '50%',

    right: 'auto',

    bottom: 'auto',

    marginRight: '-50%',

    transform: 'translate(-50%, -50%)',

  },

};

Modal.setAppElement('#root');

export default function App(){

* **State (exibição do modal)**

# Aula - Quinta - 05/09/24 – Setembro

### React Router

Faz a o gerenciamento de navegação entre as páginas

Observar a versão, sempre atualiza.

* **Install**

npm install react-router-dom

* **Site**

<https://reactrouter.com/en/main/start/tutorial>

# Aula - Terça - 10/09/24 - Setembro

### ENTREGA TP3 (16/09/24)

## TP3 React – Estrutura

Exercicio 01

# Aula - Quinta - 12/09/24 – Setembro

# Aula - Terça - 17/09/24 – Setembro

# Aula - Quinta - 19/09/24 - Setembro

# Aula - Terça - 24/09/24 – Setembro

### ENTREGA ASSESMENT

# Aula - Quinta - 26/08/24 – Setembro