<LAB365>

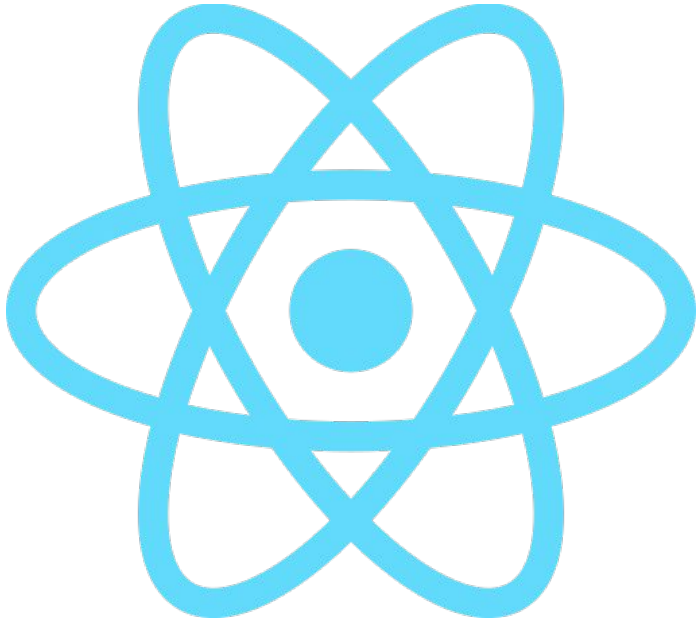
React

AGENDA | M1S08 - A1

- O que é React?
- Preparação de ambiente
- Componente Funcionais e reutilizáveis
- JSX

REACT

React é uma **biblioteca JavaScript** popular para construção de **interfaces de usuário** (UI), desenvolvida pelo Facebook. Ele permite criar componentes reutilizáveis que atualizam eficientemente quando seus dados mudam.



Principais Características

- **Baseado em Componentes**: Você constrói sua aplicação como uma árvore de componentes independentes.
- **Declarativo**: Você descreve como a UI deve aparecer em qualquer estado, e o React cuida das atualizações.
- **Virtual DOM**: Otimiza as atualizações na tela para melhor performance.
- **JSX**: Sintaxe que permite escrever HTML dentro do JavaScript.

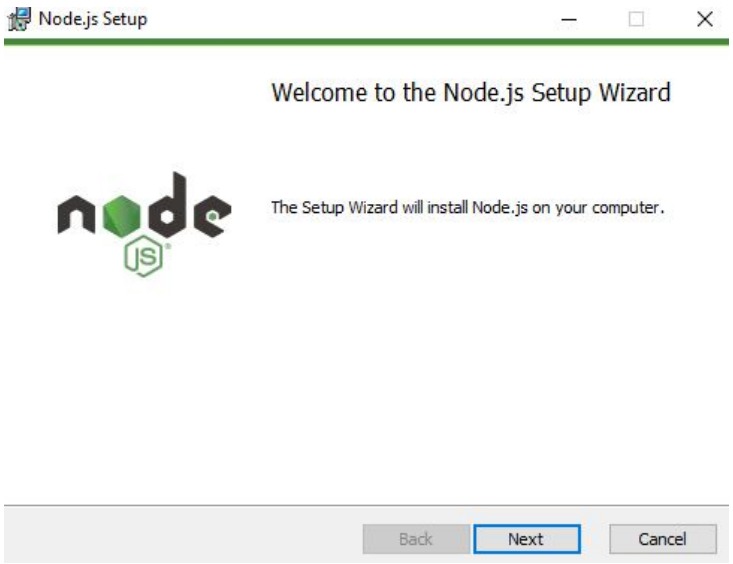
REACT: PREPARAÇÃO DE AMBIENTE

Estaremos usando o VSCode para nosso desenvolvimento em React, como essa foi a IDE que usamos até o momento temos de instalar o **Node.Js**.

https://nodejs.org/en/download

Também precisamos do **npm**, mas o mesmo vem junto na instalação do Node.

Para verificar a instalação podemos usar o seguinte comandos no cmd:
node -v
npm -v



REACT: PREPARAÇÃO DE AMBIENTE

Antes de continuarmos acho importante falarmos um pouco do npm...

O npm (Node Package Manager) é o **gerenciador de pacotes** padrão para o Node.js e um dos mais importantes ecossistemas de código aberto do mundo. Ele permite que desenvolvedores **instalem, compartilhem e gerenciem dependências** em projetos JavaScript/Node.js.

Principais Funções do npm: Instalação de pacotes, gerenciamento de dependências, execução de scripts e publicação de pacotes.

NOSSO PRIMEIRO PROJETO REACT

Para criação de nosso primeiro projeto React primeiramente precisaremos ir na pasta específica onde nosso projeto será cria, usaremos o terminal com o comando cd caminho/da/pasta.

Aqui acho importante que para facilitar o caminho das pasta a pasta de você não deve ter caracteres especiais ()[]{}/\.

Antes de continuarmos, precisamos ver como está a política de segurança do powershell, isso talvez nos impossibilite de executar o npm.

Use o comando **Get-ExecutionPolicy** para verificar politica ativa e caso esteja bloqueando o npm, podemos usar o seguinte comando **Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser** para habilitar o acesso do npm.

NOSSO PRIMEIRO PROJETO REACT

Agora vamos seguir com o seguintes comandos:

npm create vite@latest meu-app-react --
--template react : para criar um novo projeto React com o Vite.
cd meu-app-react : para entrar na pasta do projeto.
npm install : instalação do NPM.
npm run dev: executar nosso projeto.

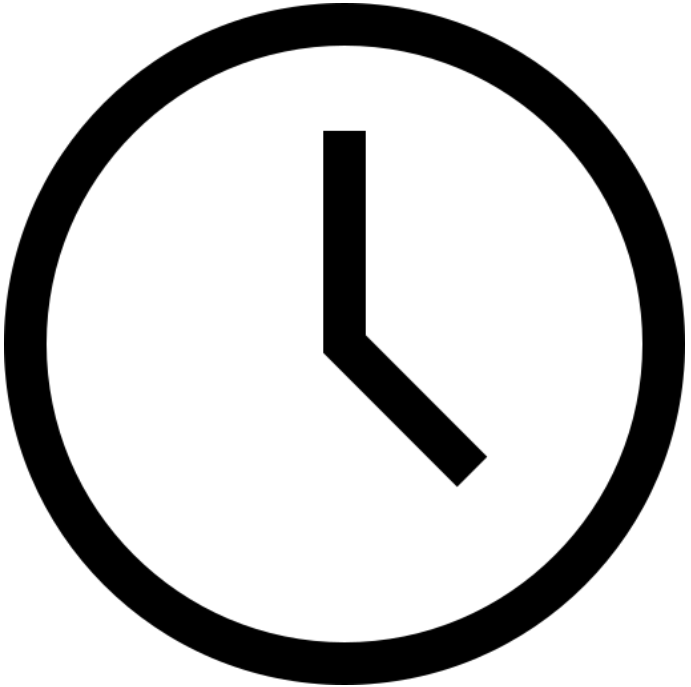
Agora podemos ver nosso projeto de exemplo!!!

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40



COMPONENTES FUNCIONAIS

Os componentes funcionais são a unidade fundamental de construção de interfaces no React. Eles representam uma evolução em relação aos componentes de classe, oferecendo uma sintaxe mais limpa e alinhada com os princípios da programação funcional.

Princípios Fundamentais:

1. Funções Puras (Quando Possível)
2. Atomicidade
3. Composição

COMPONENTES FUNCIONAIS

Funções Puras (Quando Possível)

- Idealmente, seus componentes devem ser funções puras, ou seja, dados os mesmos inputs (props), sempre retornam o mesmo JSX
- Isso facilita testes e previsibilidade

Atomicidade

- Cada componente deve ter uma única responsabilidade
- Componentes pequenos são mais fáceis de manter e reutilizar

Composição

- Componentes devem ser combinados como blocos de Lego
- Exemplo: Um componente Page composto por Header, Content e Footer

CRIANDO NOSSO PRIMEIRO COMPONENTE

Para iniciar vamos criar uma pasta components em nosso projeto, dentro dela teremos **todos os nosso componentes criados para utilização** em nosso arquivo principal (App.jsx).

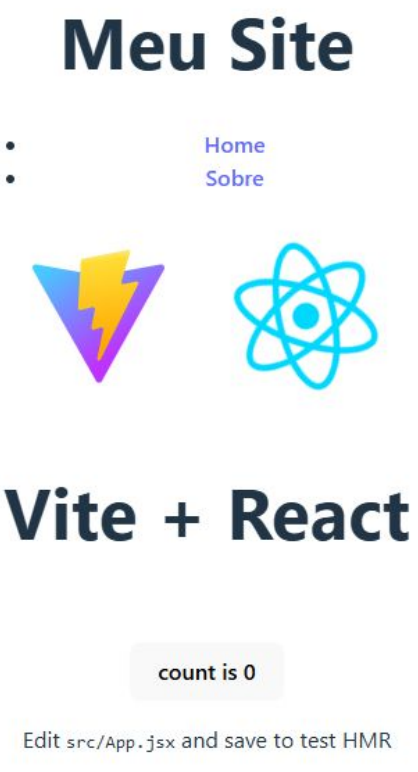
Vamos criar nosso componente, nesse caso é interessante colocarmos o nome do arquivo como uma referência à sua componentização, iremos fazer um componente de Header, então esse será o nome do nosso arquivo e também da função dentro do mesmo.

```
function Header() {
  return (
    <header>
      <h1>Meu Site</h1>
      <nav>
        <ul>
          <li><a href="/">Home</a></li>
          <li><a href="/sobre">Sobre</a></li>
        </ul>
      </nav>
    </header>
  );
}

export default Header;
```

CRIANDO NOSSO PRIMEIRO COMPONENTE

Também precisamos disponibilizar esse componente para utilização e para isso temos de fazer sua importação e utilizar a tag <Header /> para utilizar o mesmo.

```
import viteLogo from '/vite.svg'
import './App.css'
import Header from './components/header'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <Header />
```



JSX

JSX (JavaScript XML) não é um template engine, não é HTML, e não é uma string. É uma extensão sintática do JavaScript que permite descrever interfaces de usuário de forma declarativa.

Vamos ver agora as principais diferenças entre a utilização de HTML e JSX...

Característica	HTML	JSX
Atributo `class`	`class="container"`	`className="container"`
Atributo `for`	`for="email"`	`htmlFor="email"`
Estilos inline	`style="color: red;"`	`style={{ color: "red" }}`
Eventos	`onclick="handleClick()"`	`onClick={handleClick}`
Tags auto-fechadas	`<img>` ou `<input>`	`<img />` ou `<input />`
Comentários	`<!-- Comentário -->`	`{/* Comentário */}`
Valores booleanos	`disabled="true"`	`disabled={true}`

EXEMPLO

```jsx
import './Botoes.css';

function BotaoCancelar() {
  const inlineStyle = {
    margin: '10px'
  };

  return (
    <button
      className="botao-cancelar"
      style={inlineStyle}
      onClick={() => alert('Cancelado!')}
    >
      Cancelar
    </button>
  );
}

export default BotaoCancelar;
```

```css
.botao-cancelar {
    background-color: #b14646;
    border: none;
    color: white;
    padding: 12px 24px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    border-radius: 8px;
    cursor: pointer;
    transition: background-color 0.3s;
}

.botao-cancelar:hover {
    background-color: #772c2c;
}

.botao-cancelar:active {
    transform: scale(0.98);
}
```

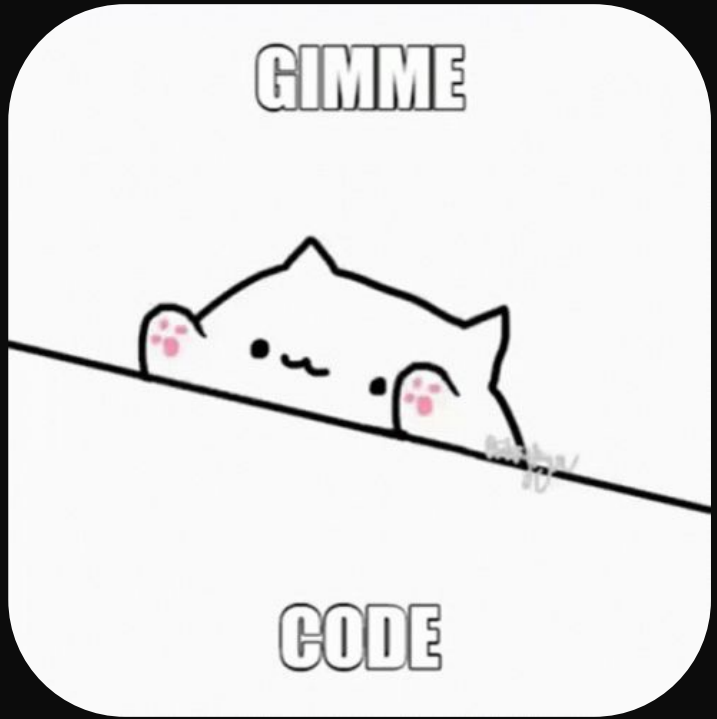
TREINANDO NOSSAS HABILIDADES!

Para iniciar nossas atividade, vamos criar um componente de Página. Será um componente que usaremos de base para páginas HTML, dentro dele usando outros componentes.

Nosso componentes serão Header, Content e Footer, todo dentro de um BasePage.

No Header teremos um componente de menu, com duas opções iniciais “Inicial” e “Contato”.

Coloque dentro do Content uma imagem e do Footer um <p> com texto de rodapé.



<LAB365>

<LAB365>

PROPS

AGENDA | M1S08 - A2

- Props
- Tipagem de dados
  - PropTypes
  - TypeScript
- Ciclo de vida de componentes
- Fluxo de dados

PROPS

Props (abreviação de properties) são um mecanismo fundamental no React para **passar dados** de um componente pai para um componente filho. Eles são imutáveis (não podem ser modificados pelo componente filho) e permitem a comunicação entre componentes.

Características

- **Unidirecionais**: Fluxo de dados apenas do pai para o filho.
- **Somente leitura**: O componente filho não pode alterar suas props.
- **Podem ser qualquer tipo**: Strings, números, arrays, objetos, funções, etc.

```
function Pessoa(props) {
  return (
    <div>
      <p>Nome: {props.nome}</p>
      <p>Idade: {props.idade}</p>
    </div>
  );
}

export default Pessoa;
```

```
<Pessoa nome="Otto" idade={33}/>
```

PROPS

Outra possibilidade que temos é trabalhar com a desconstrução do props.

Essa seria a sintaxe recomendada e dará melhor visibilidade aos dados esperados.

```
function Pessoa({nome, idade}) {
  return (
    <div>
      <p>Nome: {nome}</p>
      <p>Idade: {idade}</p>
    </div>
  );
}

export default Pessoa;
```

PROPTYPES

Para garantir que as props recebidas sejam do tipo esperado, usamos PropTypes. Isso ajuda a:

- Evitar bugs por tipos incorretos.
- Documentar o componente claramente.
- Alertar em desenvolvimento se props inválidas forem passadas.

Para usar o proptypes temos de instalar seu pacote (que atualmente não é nativo do react), usando o comando npm install prop-types.

```
<br></br>
<b>Hobbies</b>
  {hobbies.map((hobby, index) => (
    <p key={index}>{hobby}</p>
  ))}
```

```
Pessoa.propTypes = {
  nome: PropTypes.string.isRequired,
  idade: PropTypes.number,
  hobbies: PropTypes.arrayOf(PropTypes.string),
}
```

TIPAGEM COM TYPESCRIPT

TypeScript é o que chamamos de "**superset**" de JavaScript - pense nele como **uma versão ampliada e mais poderosa** da linguagem que já conhecemos. Um superset **mantém toda a funcionalidade original** (todo código JavaScript válido é também TypeScript válido), mas **adiciona novas capacidades**, no caso, um sistema de tipos sofisticado.

TypeScript representa um salto significativo em relação ao PropTypes, oferecendo uma abordagem mais robusta e profissional para o desenvolvimento com React. Enquanto o PropTypes faz verificações básicas de tipos durante a execução do código no navegador, o TypeScript eleva essa experiência para um novo patamar, realizando validações durante o desenvolvimento, antes mesmo do código ser executado.

TIPAGEM COM TYPESCRIPT

Vantagens da Migração

- Validação em Tempo de Compilação
  - PropTypes: Verifica tipos apenas em runtime (no navegador)
  - TypeScript: Detecta erros antes de executar o código (durante o desenvolvimento)
- Autocompletar Inteligente
  - Sugere propriedades disponíveis e seus tipos em IDEs como VSCode
- Tipos Mais Ricos
  - Suporte a interfaces, genéricos, tipos utilitários e muito mais
- Documentação Automática
  - Os tipos servem como documentação viva do componente
- Melhor Escalabilidade
  - Ideal para projetos grandes e equipes numerosas
- Integração com Ecossistema
  - Funciona perfeitamente com hooks, context API, Redux, etc.
- Performance
  - Nenhum overhead em produção (os tipos são removidos no build)

TIPAGEM COM TYPESCRIPT

Para utilizarmos os tipos do typescript é aconselhável que nosso projeto gerado pela vite seja em **typescript** e não em javascript.

Aqui agora geramos outro projeto, escolhendo a opção de typescript.

```tsx
interface ButtonProps {
    text: string;
    onClick: () => void;
}

function Button({ text, onClick }: ButtonProps) {
    return <button onClick={onClick}>{text}</button>;
}

export default Button;
```

```tsx
<Button text="Clique" onClick={() => console.log('OK')} />
```

```
Type 'number' is not assignable to type 'string'. ts(2322)
Button.tsx(4, 5): The expected type comes from property 'text' which is declared here on type 'IntrinsicAttributes & ButtonProps'
(property) ButtonProps.text: string
View Problem (Alt+F8)    No quick fixes available
```

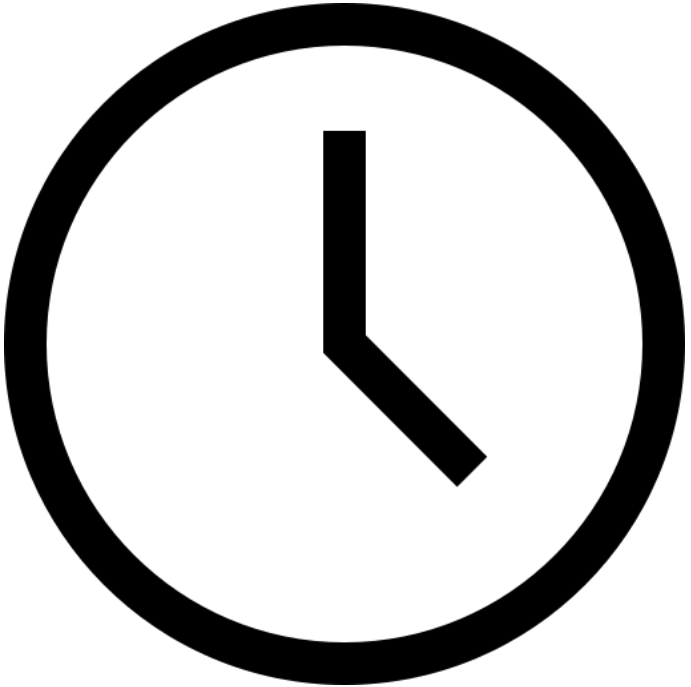
```tsx
function A
  const [c

  return (
    <>
      <Button text={32} onClick={() => console.log('OK')} />
```

INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:30
Retorno: 20:50



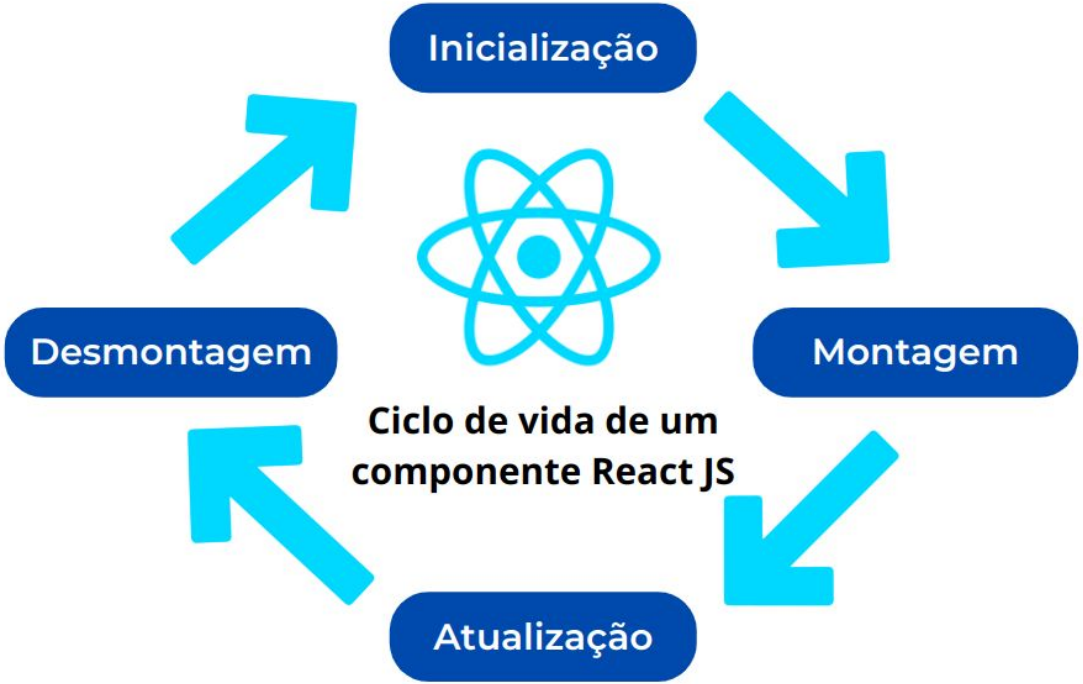
CICLO DE VIDA

O React g**erência componentes por meio de um ciclo de vida** (para componentes de classe) e um fluxo de dados unidirecional. Com o advento dos Hooks, a abordagem mudou, mas os princípios fundamentais permanecem.

Principais fases

- **Montagem**: É chamado quando o componente é criado e inserido no DOM.
  - Métodos principais: constructor, render e componentDidMount.
- **Atualização**: É chamado quando props ou state mudam.
  - Métodos principais: shouldComponentUpdate, render e componentDidUpdate.
- **Desmontagem**: É chamado antes do componente ser removido do DOM.
  - Métodos principais: componentWillUnmount.

CICLO DE VIDA



FLUXO DE DADOS

No React, o fluxo de dados segue apenas uma direção: de componentes pais para componentes filhos, através de props.

Acabamos de entender sobre props e já sabemos como os mesmos são utilizados para passagem de dados de pai para filho, mas e quando necessitamos fazer o inverso?

Nesse caso usamos callbacks que são funções passadas como props para que componentes filhos possam comunicar-se com seus pais. Funcionam como um "telefone" que o filho usa para avisar o pai sobre algo.

Como Funcionam os Callbacks?

- **Definição**: O componente pai cria uma função comum (não precisa de estado)
- **Passagem**: A função é passada para o filho como prop
- **Chamada**: O filho executa a função quando um evento ocorre (como onClick)
- **Ação**: A função original do pai é executada

FLUXO DE DADOS

```
const handleClick = () => {
  console.log("O filho foi clicado!");
  // Aqui você poderia chamar uma API ou executar outra lógica
};

return (
  <>

    <Button text="Clique" onClick={handleClick} />
```

```
function Button({ text, onClick }: ButtonProps) {
  return <button onClick={onClick}>{text}</button>;
}
```

CALLBACKS: DEFINIÇÃO

Callbacks são funções passadas como argumentos para outros componentes ou funções, que serão executadas em um momento posterior, geralmente em resposta a algum evento ou condição. No React, callbacks são usados principalmente para:

- Comunicação entre componentes (filho → pai)
- Execução de código assíncrono
- Personalização de comportamento de componentes

Pense em um callback como um número de telefone que você dá a alguém com a instrução: "Me ligue quando X acontecer". No React, o "X" seria um clique do usuário, o carregamento de dados, etc.

TREINANDO NOSSAS HABILIDADES!

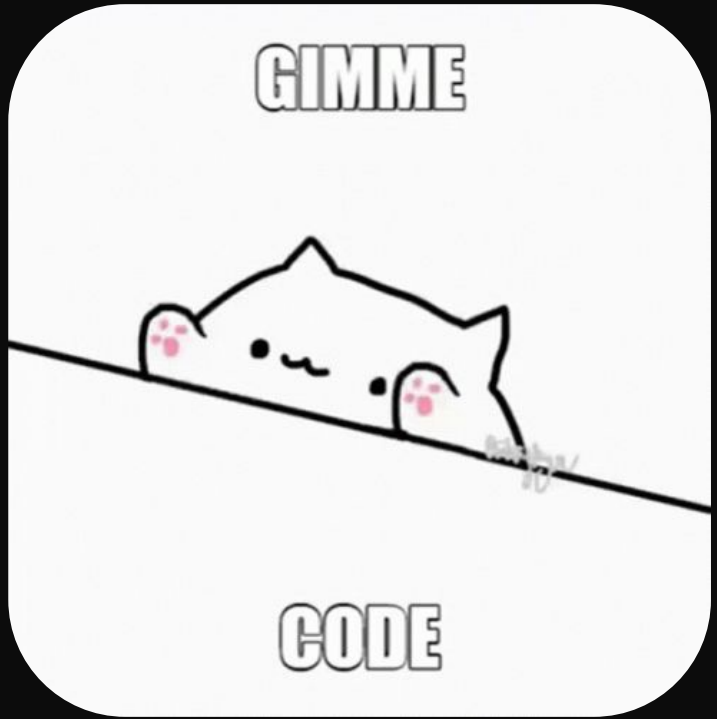
Vamos criar componentes que farão o visual de uma página de Postagem, com comentários e adição de novo comentários.

Teremos a seguinte hierarquia de componentes:

- POSTAGEM
  - COMENTÁRIOS
  - NOVO COMENTÁRIO

Estruture os componente da melhor forma possível, tendo em mente os conteúdos que vimos até o momento.

Até aqui se preocupem com o design visual e não com a usabilidade do processo.



<LAB365>

<LAB365>

REVISÃO

<LAB365> SENAI

AGENDA | M1S08 - A3

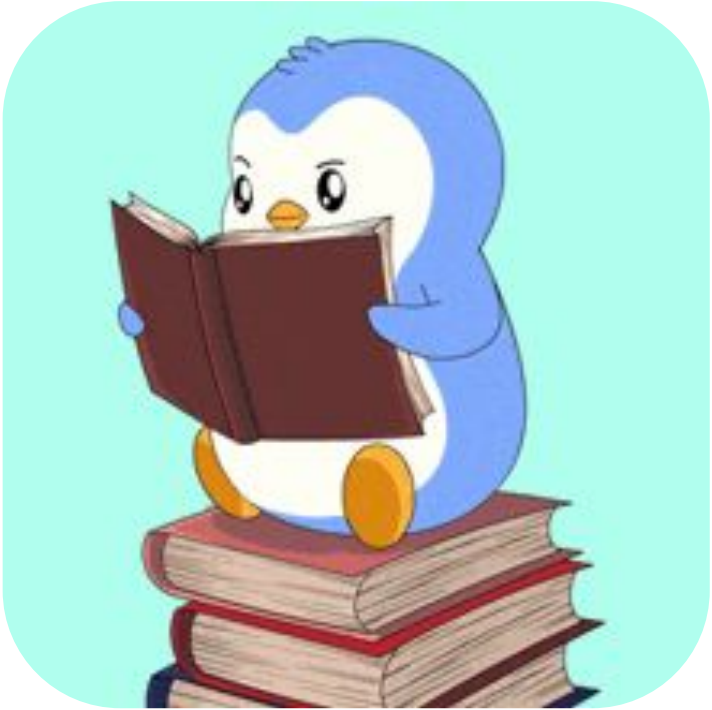
- Revisão Teórica
- Atividade de treino
- Vamos codar!

REVISÃO TEÓRICA

Faremos uma breve revisão sobre todo o conteúdo até o momento.

- Criação de projetos
- Componentes Funcionais
- Props
- TypeScript: Validação de tipos
- Ciclo de vida
- Fluxo de dados

E é isso!

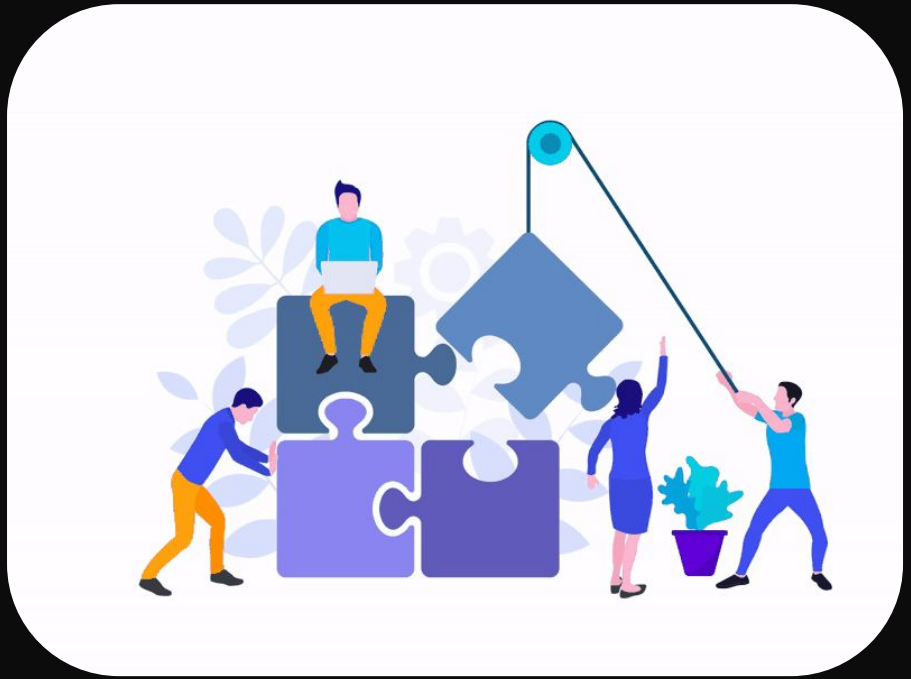


TREINANDO NOSSAS EM EQUIPE!

Desta vez vamos nos separar em equipes de focar na criação, de uma forma individual, da componentização para uma tela exibição de usuários.

- Dados do usuário
- Área de cônjuge
- Área de dependentes

A ideia é discutir como fazer essa solução em grupo, mas cada um desenvolver seu próprio código.

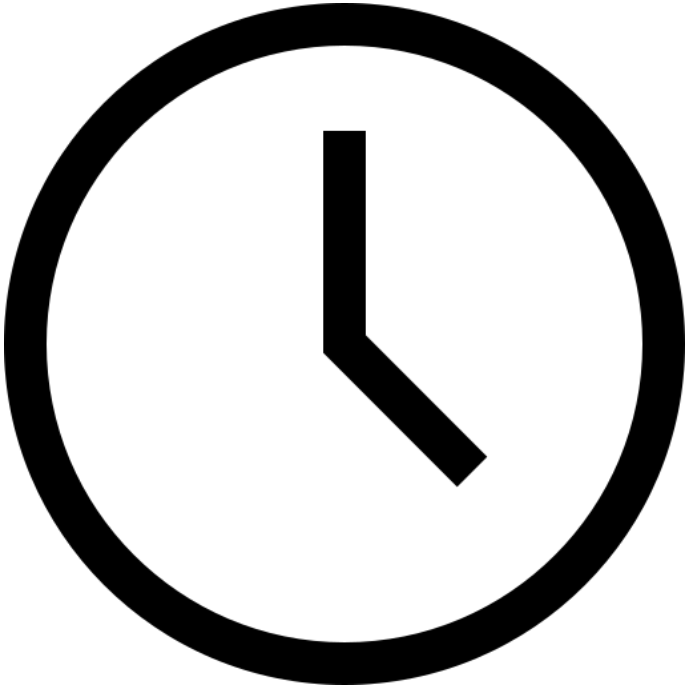


INTERVALO!

Finalizamos o nosso primeiro período de hoje. Que tal descansar um pouco?!

Nos vemos em 20 minutos.

Início: 20:20
Retorno: 20:40



VAMOS CODAR!!!

AVALIAÇÃO DOCENTE

O que você está achando das minhas aulas neste conteúdo?

Clique aqui ou escaneie o QRCode ao lado para avaliar minha aula.

Sinta-se à vontade para fornecer uma avaliação sempre que achar necessário.

<LAB365>