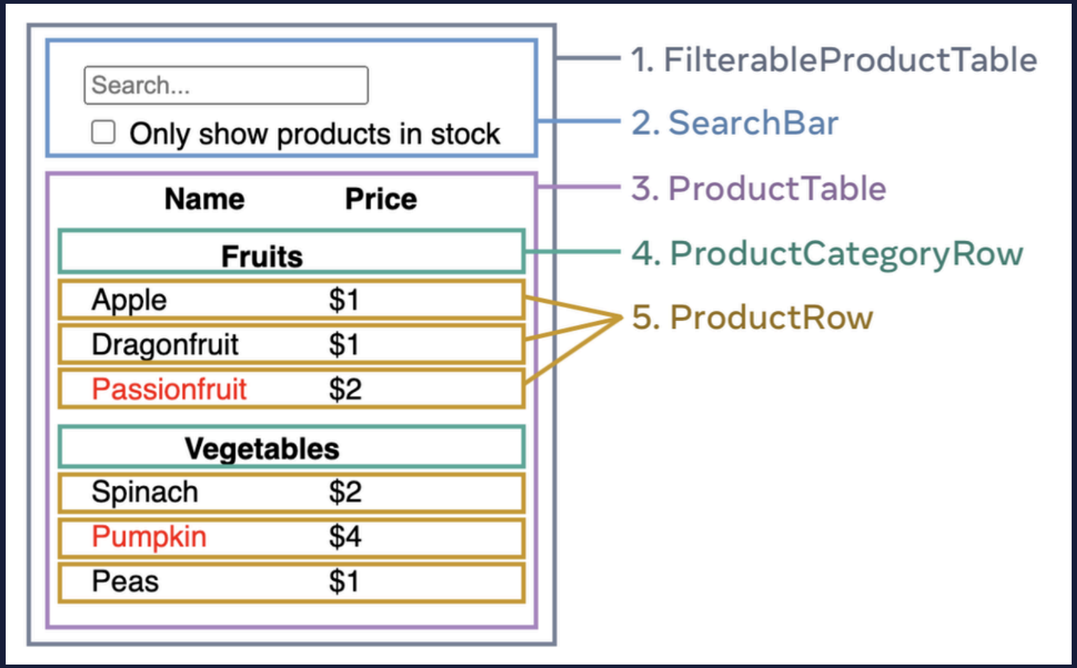
COMPONENTES COM REACT JS

Victor Angelo Marcorin
github.com/victoramccomp

Uso do ReactJS em 2025

O que é um Componente?

É a unidade básica de uma interface React. Pense nele como uma "função" que recebe dados e retorna HTML (JSX).



Anatomia de um Componente

Um Componente é composto por Lógica (Javascript/Hooks), Estrutura (JSX) e Estilo (CSS).

O componente é independente. Ele deve funcionar sozinho se receber os dados corretos.

```jsx
// Definição do Componente
const UserCard = ({ name, role }) => {

  // 1. Lógica (Hooks e Funções)
  const [isFollowed, setIsFollowed] = useState(false);

  const handleFollow = () => {
    setIsFollowed(!isFollowed);
  };

  // 2. Estrutura (JSX / HTML-like)
  return (
    <div className="card-container">
      <h3>{name}</h3>
      <p>{role}</p>

      <button onClick={handleFollow}>
        {isFollowed ? 'Seguindo' : 'Seguir'}
      </button>
    </div>
  );
};

export default UserCard;
```

Props - A Comunicação de Dados

São argumentos passados de um componente pai para um filho.
São imutáveis.

O componente que recebe a Prop não pode alterá-la, apenas usá-la para renderizar algo ou tomar decisões.

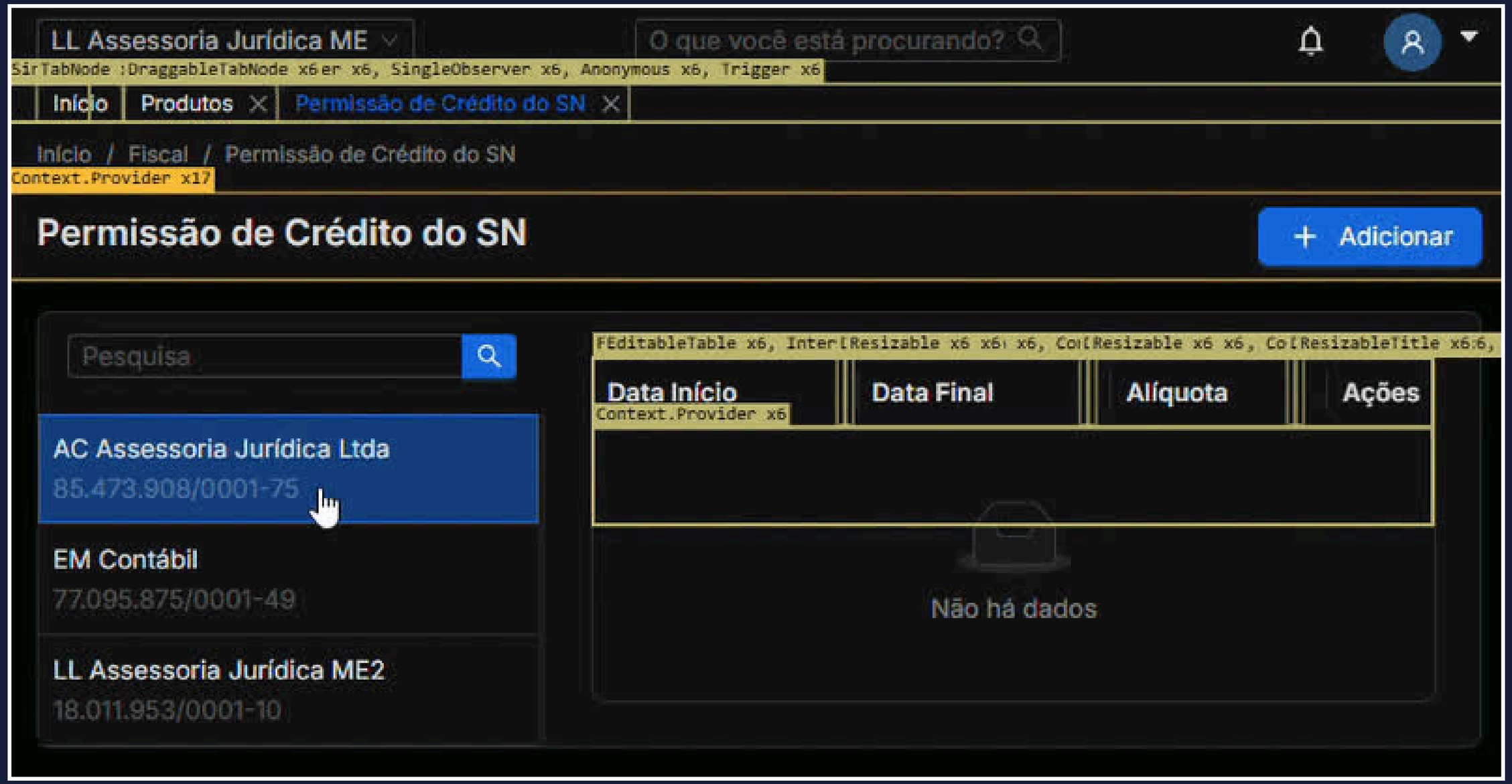
```tsx
// 1. Definição do "Contrato" (Interface)
interface ButtonProps {
  label: string;           // Obrigatório
  color?: 'blue' | 'red';  // Opcional (Union Type)
  onClick: () => void;     // Callback (Evento que sobe para o pai)
}

// 2. O Componente recebe as Props e as desestrutura
const CustomButton: React.FC<ButtonProps> = ({ label, color = 'blue', onClick }) =>
  return (
    <button
      style={{ backgroundColor: color, color: 'white', padding: '10px' }}
      onClick={onClick}
    >
      {label}
    </button>
  );
};

export default CustomButton;
```

State - A Memória do Componente

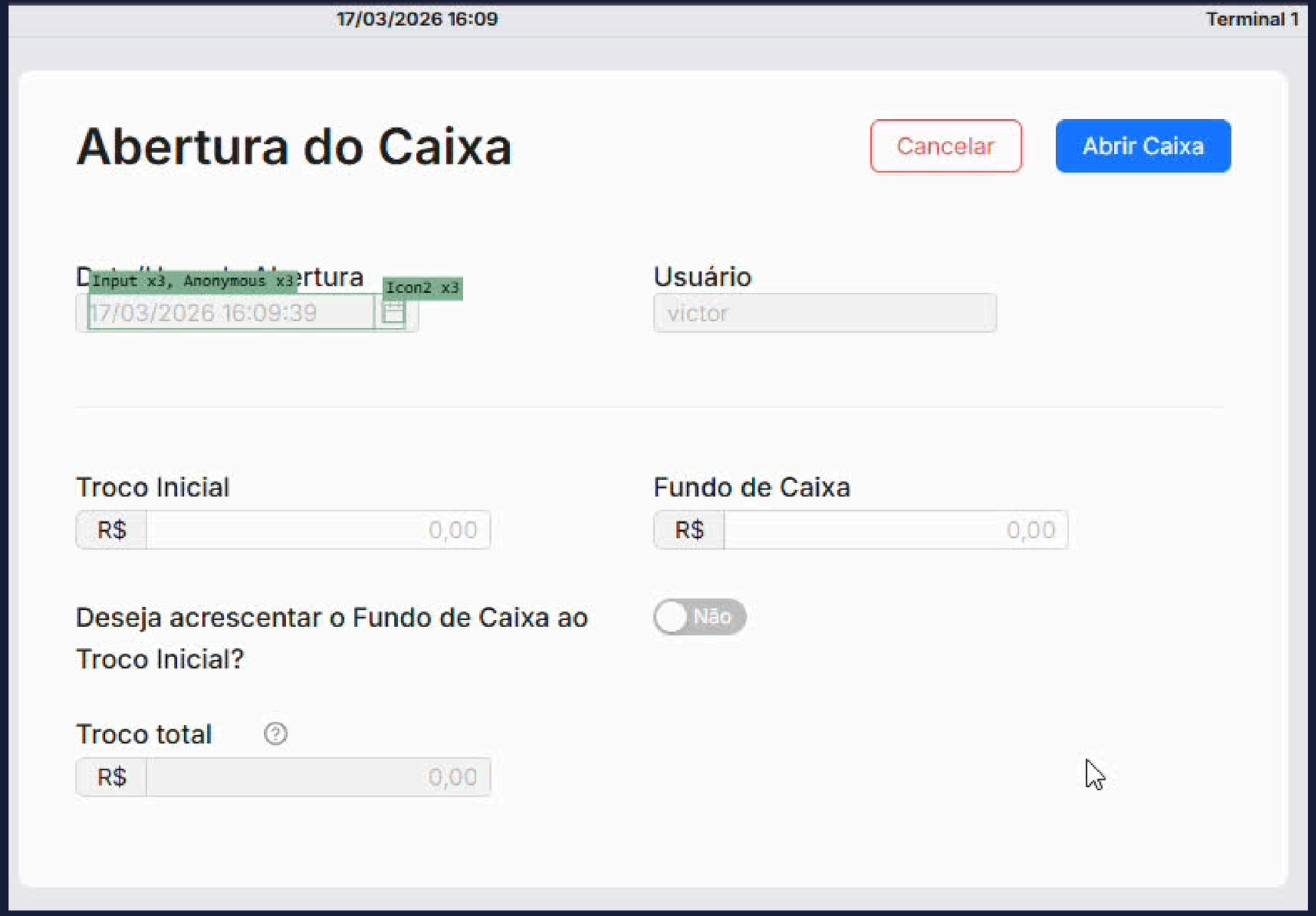
É o estado interno que pode mudar ao longo do tempo. Por exemplo, um campo de texto ou um carregamento.

Sempre que o State muda, o React "re-renderiza" o componente automaticamente para atualizar as informações da tela.

State - A Memória do Componente



State - A Memória do Componente



Hooks (useState e useEffect)

Funções especiais que permitem "conectar" recursos do React em componentes funcionais.

- useState: Cria e gerencia o estado.
- useEffect: chamadas de API (Busca de informação externa)
- useRef: o useRef serve para guardar um valor que persiste entre as renderizações, mas que não deve disparar uma nova pintura na tela quando for alterado.

Composição vs. Reutilização

Em vez de herança, usamos composição. Criamos componentes pequenos e genéricos para montar estruturas complexas.

Evitamos códigos de componentes que podem se tornar Monólitos, ao invés quebramos para que possa ser mais genérico e reutilizável.

Top 4 Componentes

Filters

- Encapsula todo o comportamento de “drawer de filtros + formulário + botões de rodapé”.
- Usa composição (Filters.Container + Filters.Item), então as páginas só declaram os campos, não a infraestrutura.
- Esconde detalhes de implementação como campos invisíveis de operador, filtros padrão e comportamento de limpar.

```
<Filters.Container form={form} open>
  <Filters.Item
    label="Status"
    name="Status"
    operator="Equal"
  >
    <Input />
  </Filters.Item>
</Filters.Container>
```

Top 4 Componentes

FModal

Envolve o Modal do Ant Design com:

- Presets de tamanho (default, small, large, mini).
- API imperativa (open, close, disableSaveButton, isOpen).
- UX consistente.

Evita que cada página reimplemente esse comportamento.

```
<FModal
  className={containerClassName}
  ref={ref}
  onCancel={() => {
    closeModal();
  }}
  title="Atribuir CEST"
  footer={null}
>
  <Space.Compact style={{ marginBottom: 16 }}>···
  </Space.Compact>
  <FTable<DataType>
    dataSource={dataSource}···
    columns={columns}
  />
</FModal>
```

Top 4 Componentes

FSelect

Genérico sobre DataType, funciona com qualquer service.

Centraliza:

- Fetch de dados (useList),
- Busca client-side,
- Mapeamento label/value,
- Comportamento de retorno (id vs objeto completo).

Usa optionsFn para permitir pós-processamento de opções em cada uso.

```
<Form.Item
  label="Módulos"
  name="modulos"
  rules={[required]}
>
  <FSelect
    service={serviceModulos}
    labelProp="nome"
    valueProp="id"
    size="small"
    mode="tags"
    allowClear
    className="select"
    placeholder="Selecione os Módulos"
  />
</Form.Item>
```

Top 4 Componentes

FEditableTable

Encapsula um padrão de UX complexo (edição por linha, com ações salvar/cancelar/editar/remover).

Esconde a lógica de:

- cabeçalho redimensionável,
- seleção de linhas,
- estados de edição na linha
- sincronização com Form e localData.

Tipagem forte (DataType, DataTypeParent, dataSourcePropName).

```tsx
<Form.List name={["permissaoCreditoSimplesNacional"]}>
  {(fields) => (
    <>
      <FEditableTable<DataTypePermissaoCreditoSimplesNacional, PessoaJuridica>
        dataSourcePropName="permissaoCreditoSimplesNacional"
        dataSource={fields}
        actions={{
          save: saveRegister,
          remove: removeRegister
        }}
        enableSaveButton={enableSaveButton}
        initialValues={servicePermissaoCreditoSimplesNacional.initialValues}
        ref={tableRef}
        pagination={false}
        resizable
        columns={columns(form)}
        localData={localData}
      />
    </>
  )}
</Form.List>
```

Dúvidas?

Fontes

State of JS 2025: https://2025.stateofjs.com/en-US
ReactJS: https://react.dev/learn/thinking-in-react
Patterns React: https://www.patterns.dev/react
Link Canva da Apresentação