🚗 Atividade Guiada — Simulador de Veículo com React Hooks

🎯 Objetivo da Atividade

Nesta atividade, você vai aprender a usar **React Hooks** (`useState` e `useEffect`) criando um **simulador de veículo**.

Você vai:

- Controlar o estado do motor e velocidade;
- Reagir automaticamente às mudanças de estado;
- Aplicar lógica condicional para simular situações reais de um carro.

🧠 Antes de começar: Entendendo os Hooks

◇ useState

O `useState` permite criar **variáveis de estado** dentro de um componente funcional.

Sintaxe:

```
const [estado, setEstado] = useState(valorInicial);
```

- `estado` → valor atual da variável
- `setEstado` → função para atualizar o valor
- `valorInicial` → valor inicial da variável

Exemplo:

```
const [contador, setContador] = useState(0);
setContador(contador + 1);
```

◇ useEffect

O `useEffect` permite executar **efeitos colaterais** em resposta a mudanças no estado.

Sintaxe:

```
useEffect(() => {
  // código executado quando variáveis mudam
}, [variaveisMonitoradas]);
```

- `[variaveisMonitoradas]` → lista de variáveis que dispararão o efeito
- `[]` → executa apenas **uma vez**, ao montar o componente

Exemplo:

```
useEffect(() => {
  console.log("O componente foi renderizado!");
}, []);
```

⚙️ Parte 1 — Criando o projeto React

1️⃣ Crie o projeto com Vite:

```
npm create vite@latest .
npm install
npm run dev
```

2️⃣ Estrutura de pastas sugerida:

```
src/
 ├ componentes/
 │ └ Veiculo/
 │   ├ Veiculo.jsx
 │   └ Veiculo.module.css
 └ App.jsx
```

🚗 Parte 2 — Contexto do Componente

O componente `Veiculo` deve:

- Ligar/desligar o motor;
- Acelerar/frear;
- Exibir velocidade e combustível;
- Reagir automaticamente às mudanças (como alertas e logs).

🧩 Parte 3 — Código Comentado (`Veiculo.jsx`)

```
import { useState, useEffect } from "react";
import styles from "./Veiculo.module.css";

export default function Veiculo() {
```

```
// 1️⃣ Estado do motor
const [ligado, setLigado] = useState(false);

// 2️⃣ Estado da velocidade
const [velocidade, setVelocidade] = useState(0);

// 3️⃣ Estado do combustível
const [combustivel, setCombustivel] = useState(100);

// 4️⃣ Efeito para velocidade
useEffect(() => {
  if (ligado) {
    console.log(`Velocidade atual: ${velocidade} km/h`);

    if (velocidade === 100) {
      alert("🚀 Cuidado! Alta velocidade!");
    }
  }
}, [velocidade, ligado]);

// 5️⃣ Efeito para ligar/desligar
useEffect(() => {
  if (ligado) {
    console.log("🚗 O carro foi ligado!");
  } else {
    console.log("🛑 O carro foi desligado!");
    setVelocidade(0);
    alert("🛑 O carro foi desligado!");
  }
}, [ligado]);

// 6️⃣ Função ligar/desligar
function ligarDesligar() {
  if (!ligado && combustivel <= 0) {
    alert("⛽ Sem combustível! Abasteça antes de ligar.");
    return;
  }
  setLigado(!ligado);
}

// 7️⃣ Função acelerar
function acelerar() {
  if (!ligado){
    return;
  }
  if (combustivel > 0) {
    setVelocidade(velocidade + 10);
    setCombustivel(Math.max(combustivel - 5, 0));
  } else {
    alert("⛽ Acabou o combustível!");
    setLigado(false);
  }
}
```

```
  // 8 Função frear
  function frear() {
    if (!ligado) return;

    if (velocidade > 0) {
      setVelocidade(velocidade - 10);
      setCombustivel(Math.min(combustivel + 1, 100));
    }
  }

  // 9 JSX
  return (
    <>
      <div className={styles.painel}>
        <h2>Painel do Veículo</h2>
        <p><strong>Status:</strong> {ligado ? "🚗 Ligado" : "🔘 Desligado"}</p>
        <p><strong>Velocidade:</strong> {velocidade} km/h</p>
        <p><strong>Combustível:</strong> {combustivel.toFixed(0)}%</p>

        <div className={styles.botoes}>
          <button onClick={ligarDesligar}>
            {ligado ? "Desligar" : "Ligar"}
          </button>
          <button onClick={acelerar} disabled={!ligado}>Acelerar</button>
          <button onClick={frear} disabled={!ligado || velocidade ===
0}>Frear</button>
        </div>
      </div>
    </>
  );
}
```

🎨 Parte 4 — Estilo (`Veiculo.module.css`)

```
.painel {
  background-color: #f5f5f5;
  border-radius: 0.75rem;
  padding: 1.25rem;
  text-align: center;
  width: 18.75rem;
  margin: 2.5rem auto;
  box-shadow: 0 0.25rem 0.5rem rgba(0,0,0,0.2);
}

.botoes {
  display: flex;
  justify-content: center;
  gap: 0.625rem;
  margin-top: 1rem;
}
```

```css
button {
  background-color: #007bff;
  border: none;
  color: white;
  padding: 0.5rem 0.875rem;
  border-radius: 0.375rem;
  cursor: pointer;
  font-size: 0.875rem;
}

button:disabled {
  background-color: gray;
  cursor: not-allowed;
}
```

🚀 Parte 5 — Usando o componente no `App.jsx`

```jsx
import Veiculo from "./componentes/Veiculo/Veiculo";

function App() {
  return (
    <>
        <h1>Simulador de Veículo</h1>
        <Veiculo />
    </>
  );
}

export default App;
```

🧩 Parte 6 — Desafios Extras

1️⃣ Impedir que o carro ligue se o combustível estiver abaixo de 10%

Dica: Dentro da função `ligarDesligar()`, adicione uma validação condicional para verificar se o combustível é menor que 10%. Se for, exiba um alerta e use `return` para impedir a execução do restante da função.

2️⃣ Exibir alerta se o combustível estiver abaixo de 20%

Dica: Crie um novo `useEffect` que monitora o estado `combustivel`. Quando o combustível cair abaixo de 20%, exiba um alerta ao usuário. Para evitar alertas repetidos, você pode adicionar uma condição que verifica se o carro está ligado.

---

3 Criar botão "Abastecer" que enche o tanque

Dica: Crie uma nova função chamada `abastecer()` que define o combustível para 100. Adicione um novo botão no JSX que chama essa função. O botão deve ficar desabilitado quando o carro estiver ligado (só pode abastecer com o carro desligado).

---

4 Mostrar barra de combustível visual (progress bar)

Dica: Use a tag HTML `<progress>` ou crie uma div com largura dinâmica baseada no valor do combustível. Para usar `<progress>`, defina os atributos `value` e `max`.

Exemplo com `<progress>`:

```
<progress value={combustivel} max="100"></progress>
```

Exemplo com div customizada (no JSX):

```
<div className={styles.barraContainer}>
  <div
    className={styles.barraCombustivel}
    style={{ width: `${combustivel}%` }}
  ></div>
</div>
```

CSS correspondente:

```
.barraContainer {
  background-color: #ddd;
  border-radius: 0.5rem;
  height: 1.25rem;
  margin-top: 0.625rem;
  overflow: hidden;
}

.barraCombustivel {
  background-color: #28a745;
  height: 100%;
  transition: width 0.3s ease;
}
```

---

5 Bloquear aceleração acima de 120 km/h

Dica: Analise a função `acelerar()` e crie uma condição que verifica se `velocidade é maior que 120`. Você deve interromper a execução da função quando a velocidade máxima é atingida.

Dicas de estudo

- `useState` → painel de controle do componente, armazena dados que mudam ao longo do tempo;
- `useEffect` → ações automáticas quando o estado muda, perfeito para logs, alertas e sincronizações;
- Experimente modificar os valores e observar o comportamento do console e alertas;
- Use `console.log()` estrategicamente para entender o fluxo de execução do seu código;
- A unidade `rem` é relativa ao tamanho da fonte raiz (geralmente 16px), facilitando a responsividade do layout.