SOFTEX
PERNAMBUCO

Softex

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

Aula 28 | Módulo: Typescript e Orientação a Objetos (continuação) e React (continuação)

- Introdução ao teste de software e aplicação em código orientado a objetos:
  - Testes unitários com exemplos simples
  - Testes em métodos e interações entre objetos
- Ciclo de vida com useEffect: execução em diferentes fases

Caso queira estudar usando um editor na web

- Acesse:
  https://vite.new/react-ts

Typescript e Orientação a Objetos (continuação)

Configurando o ambiente para rodar o React

- Node.js (LTS recomendado >= 20):
  - **Windows/macOS:**
    - baixe o instalador LTS no site do Node.
  - **Linux (Deb/Ubuntu):**
    - curl -fsSL https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
    - nvm install --lts
    - nvm use --lts
- Confirmar instalação rodando os comandos abaixo em algum terminal:
  - **node -v**
  - **npm -v**
- Se “npm/node não reconhecido”, feche o terminal e tente pelo Command Prompt.

Typescript e Orientação a Objetos (continuação)

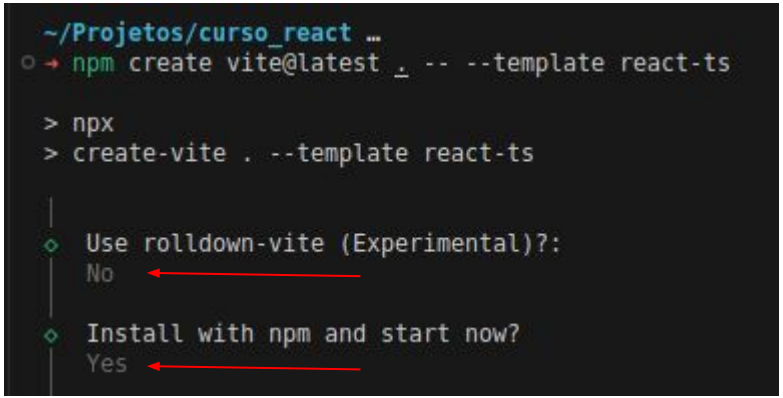
Abrindo/Criando um projeto

- Para ABRIR um projeto existente:
  - Abra o VSCode, clique em File > Open Folder… (Arquivo > Abrir Pasta…) e procurar a pasta do projeto existente.
  - No terminal, rode o comando ao lado para iniciar: **npm run dev**
- Para CRIAR um projeto novo
  - Abra o VSCode, clique em File > Open Folder… (Arquivo > Abrir Pasta…).
  - Escolha um local para criar a sua pasta, crie uma nova pasta e dê o nome de **seunome_aula_react_28**. Depois dê dois clique nessa pasta criada e clique em Selecionar pasta. O VSCode reabrirá dentro dessa pasta que foi criada.

Typescript e Orientação a Objetos (continuação)

Criar o projeto com Vite

- Criando um projeto com TypeScript:
  - **npm create vite@latest . -- --template react-ts**
- Durante a instalação do React usando o Vite, algumas opções podem surgir:



Typescript e Orientação a Objetos (continuação)

Se algo travar: portas ou permissões

- Porta ocupada (erro ao iniciar ou tela vazia):
  - Iniciar em outra porta: **npm run dev -- --port 5174**
- Windows: “npm não reconhecido”
  - Feche e reabra o PowerShell/CMD.
  - Verifique se o Node foi instalado para “All Users” (às vezes a variável de ambiente não atualiza).
- Linux/macOS: erro de permissão (EACCES)
  - mkdir -p ~/.npm-global
  - npm config set prefix ~/.npm-global
  - echo 'export PATH=$HOME/.npm-global/bin:$PATH' >> ~/.bashrc && source ~/.bashrc

O que são Testes de Software

- Testes verificam se o código funciona como o esperado.
- Ajudam a detectar erros antes que causem problemas maiores.
- Trazem confiança ao programador e à equipe.
- Reduzem retrabalho e custos no desenvolvimento.

Typescript e Orientação a Objetos (continuação)

Tipos mais comuns de testes

- **Testes Unitários:** verificam pequenas partes do código (ex: um método).
- **Testes de Integração:** verificam se partes diferentes funcionam juntas.
- **Testes de Interface:** garantem que o sistema se comporta bem visualmente.
- **Testes Manuais vs Automatizados:**
  - **Manuais:** feitos pelo usuário.
  - **Automatizados:** feitos por scripts.

Normalmente começamos sempre pelos unitários, porque são simples e garantem a base.

Typescript e Orientação a Objetos (continuação)

Criando ambiente de testes em React + TypeScript

- Precisamos instalar algumas bibliotecas:
  - **npm install -D vitest @testing-library/react @testing-library/jest-dom happy-dom**

Pacote	Função
vitest	Framework de testes (como o Jest, mas nativo do Vite)
@testing-library/react	Permite renderizar e interagir com componentes React nos testes
@testing-library/jest-dom	Adiciona comandos extras, tipo toBeInTheDocument()
happy-dom	Simula o ambiente de navegador para o React funcionar nos testes

Typescript e Orientação a Objetos (continuação)

Criando ambiente de testes em React + TypeScript

- No arquivo **package.json**, adicione:
  - "scripts": {
  - "test": "vitest"
  - }
- No arquivo **vite.config.ts**, adicione dentro do defineConfig({...}):
  - test: {
  - environment: "happy-dom",
  - globals: true,
  - setupFiles: "./src/tests/setup.ts"
  - },

Typescript e Orientação a Objetos (continuação)

Criando ambiente de testes em React + TypeScript

- Agora vamos criar a pasta **tests** dentro da pasta **src**:
  - Crie o arquivo **setup.ts** dentro da pasta tests (src/tests/setup.ts) e adicione:
    - **import "@testing-library/jest-dom";**
- Agora vamos criar um componente para testar.
- Para facilitar, deixei o componente e os arquivos de testes no repositório:
  - **https://github.com/hygorrasec/curso_react/tree/somador/src**
- Os arquivos de testes estão na pasta **tests**. Os testes só rodam se existirem arquivos que terminem com **.test.ts**, **.test.tsx**, **.spec.ts** etc.
- O componente está na pasta **components**.
- O **App.tsx** do repositório já está importando o nosso componente **Somador**.
- Depois que tiver tudo configurado, abra um novo terminal e rode o comando para testar: **npm run test**

Typescript e Orientação a Objetos (continuação)

Criando ambiente de testes em React + TypeScript

- Se tudo fluir bem, o teste será executado:

```
curso_react on  somador [!] …
→ npm run test


> curso_react@0.0.0 test
> vitest


 DEV  v4.0.8 /home/hygorrasec/Projetos/curso_react

 ✓ src/tests/Somador.test.tsx (1 test) 25ms
   ✓ Componente Somador (1)
     ✓ deve somar dois números corretamente 24ms

 Test Files  1 passed (1)
      Tests  1 passed (1)
   Start at  11:12:20
   Duration  405ms (transform 34ms, setup 57ms, collect 70ms, tests 25ms, environment 156ms
, prepare 6ms)

 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

Typescript e Orientação a Objetos (continuação)

Teste de interações entre componentes

- Um componente pode representar dois objetos interagindo entre si.
- Podemos testar se o comportamento entre eles funciona corretamente (ex: um jogador atacando outro e reduzindo sua vida).
- Isso ajuda a entender como o estado muda com base nas ações.

Typescript e Orientação a Objetos (continuação)

Exercício prático de teste de interações entre componentes

- Vamos acessar o repositório abaixo e copiar o código dos arquivos:
  - **src/App.tsx**
  - **src/components/Batalha.tsx**
  - **src/tests/Batalha.test.tsx**
- Ou faça o clone do repositório e acesse a branch “batalha”:
  - Comando: **git checkout batalha**
- Link do repositório:
  - **https://github.com/hygorrasec/curso_react/tree/batalha/src**

Typescript e Orientação a Objetos (continuação)

Teste de interações entre componentes

- Se tudo der certo, o resultado do teste será esse:

```
curso_react on  batalha [x] took 47m 45,2s …
→ npm run test


> curso_react@0.0.0 test
> vitest


 DEV  v4.0.8 /home/hygorrasec/Projetos/curso_react

 ✓ src/tests/Batalha.test.tsx (3 tests) 28ms
   ✓ Componente Batalha (3)
     ✓ deve exibir os dois jogadores com vida inicial 100 16ms
     ✓ deve reduzir a vida do vilão para 70 ao atacar uma vez 6ms
     ✓ deve mostrar mensagem de vitória quando o vilão chegar a 0 ou menos 4ms

 Test Files  1 passed (1)
      Tests  3 passed (3)
   Start at  12:00:06
   Duration  409ms (transform 34ms, setup 56ms, collect 71ms, tests 28ms, environment 157ms
, prepare 6ms)

 PASS  Waiting for file changes...
       press h to show help, press q to quit
```

React JS (continuação)

O que é o useEffect?

- O **useEffect** permite executar efeitos colaterais em componentes funcionais.
- O que são efeitos colaterais?
  - São ações que ocorrem fora do fluxo normal de renderização, como:
    - Buscar dados de uma API
    - Atualizar o título da aba
    - Criar temporizadores
    - Interagir com o localStorage

React JS (continuação)

useEffect executa quando?

O useEffect pode ser executado em três momentos diferentes:

1. **Toda renderização (executa sempre que o componente renderiza):**
   - useEffect(() => {...});
2. **Somente na montagem (executa apenas uma vez (quando o componente é criado)):**
   - useEffect(() => {...}, []);
3. **Quando algo muda (executa apenas quando o valor dentro do array muda):**
   - useEffect(() => {...}, [variável]);

Typescript e Orientação a Objetos (continuação)

Exercícios práticos

- Vamos acessar o repositório abaixo e copiar o código dos arquivos:
  - **src/App.tsx**
  - **src/components/Exemplo1.tsx**
  - **src/components/Exemplo2.tsx**
  - **src/components/Exemplo3.tsx**
- Ou faça o clone do repositório e acesse a branch “useeffect”:
  - Comando: **git checkout useeffect**
- Link do repositório:
  - **https://github.com/hygorrasec/curso_react/tree/useeffect/src**

React JS (continuação)

useEffect com limpeza (cleanup)

- Quando usamos recursos que devem ser removidos (como temporizadores ou eventos), precisamos “limpar” no final.
- Cleanup: evita vazamento de memória e execução duplicada.

Typescript e Orientação a Objetos (continuação)

Exercícios práticos

- Vamos acessar o repositório abaixo e copiar o código dos arquivos:
  - **src/App.tsx**
  - **src/components/Temporizador.tsx**
- Ou faça o clone do repositório e acesse a branch “temporizador”:
  - Comando: **git checkout temporizador**
- Link do repositório:
  - **https://github.com/hygorrasec/curso_react/tree/temporizador/src**

Typescript e Orientação a Objetos (continuação)

Observação sobre o texto “Temporizador limpo!” no console.

- O React "finge desmontar" o componente no modo de desenvolvimento.
- Quando você usa isso no seu **main.tsx**:
  - **<StrictMode>** **<----------**
  - <App />
  - **</StrictMode>** **<----------**
- O React faz o seguinte:
  - Monta o componente (executa o useEffect)
  - Desmonta o componente logo em seguida (executa o return)
  - Monta novamente o componente (executa o useEffect de novo)
- Ele faz isso de propósito para verificar se o seu efeito tem “limpeza correta”.

ATÉ A PRÓXIMA AULA!

Front-end - Design. Integração. Experiência.

Professor: Hygor Rasec
https://www.linkedin.com/in/hygorrasec
https://github.com/hygorrasec