Aplicação: Pokédex Interativa em React

Disciplina: Programação IV
Aluno: Thierry dos Santos

1.Introdução

Este documento detalha o desenvolvimento de uma Pokédex interativa reescrita na arquitetura React, utilizando HTML, CSS e JavaScript (ES6+). O projeto tem como base o consumo de dados da PokéAPI e visa aplicar o Padrão de Projeto Container/Presenter como estrutura arquitetural principal, demonstrando o uso de boas práticas de programação, componentização modular e organização eficiente de código. O resultado é uma aplicação limpa, manutenível e com clara separação entre a lógica de negócio e a camada de apresentação.

2. Padrão de Projeto Utilizado: Container/Presenter

O Padrão Container/Presenter (ou Container/View) foi adotado para maximizar a modularidade e a clareza do código. Este padrão estabelece uma divisão rigorosa de responsabilidades, fundamental para projetos que envolvem gerenciamento de estado complexo e chamadas assíncronas de API.

2.1. Justificativa e Contexto da Escolha

A escolha deste padrão foi motivada pela natureza da aplicação, que possui uma forte dependência de dados externos e uma interface dinâmica.

1. **Separação de Preocupações (SoC):** O padrão isola a lógica de onde os dados vêm e como eles são manipulados (Container) da lógica de como os dados são exibidos (Presenter). Isso garante que a interface do usuário (UI) possa ser alterada sem impactar a lógica de busca de dados, e vice-versa.

2. **Facilidade de Teste:** Os Componentes Presentational tornam-se "puros", pois dependem apenas das props que recebem. Isso simplifica a escrita de testes unitários, pois não é necessário simular chamadas de API ou estados complexos para verificar a renderização da UI.

2.2. Aplicação Prática no Sistema

A implementação seguiu a seguinte divisão de componentes:

1. A. O Container: PokedexContainer.jsx

O PokedexContainer é o componente "inteligente", atuando como o controlador principal da aplicação. Suas responsabilidades incluem:

- **Gerenciamento de Estado:** Utiliza useState para manter o estado global da aplicação (ex: pokemonData, currentPokemonId, isShiny, isLoading).

- **Comunicação com API:** A função fetchPokemon (envolvida por useCallback para otimização) reside aqui, sendo responsável por buscar os dados do Pokémon na PokéAPI e atualizar o estado.

- **Distribuição de Dados:** Ele coordena a renderização, passando os dados processados e as funções de manipulação de estado (callbacks) para os componentes Presentational via props.

- **Exemplo:** Passa a função handleSearch para o componente SearchBar e o objeto pokemonData para o PokemonCard.

2. B. Os Presenters (Views): PokemonCard.jsx, StatBar.jsx, etc.

Os Presenters são os componentes "burros" ou "de exibição". Eles se concentram exclusivamente em como o Pokémon é visto:

- **PokemonCard.jsx:** Recebe o objeto completo do Pokémon, o booleano isShiny e o handler onShinyToggle via props. Ele é responsável por renderizar a imagem, o nome, os tipos e, crucialmente, compor os componentes internos como o StatBar.

- **SearchBar.jsx:** Recebe a função onSearch via props e gerencia apenas o estado local do campo de input, repassando o termo de busca para o Container.

- **StatBar.jsx:** Componente de UI pura, que recebe statName e statValue e renderiza a barra de progresso visual correspondente.

3. Componentização e Boas Práticas de Programação

A arquitetura em React permitiu a aplicação de diversas boas práticas, essenciais para a manutenibilidade do projeto.

3.1. Reutilização de Código (DRY Principle)

O princípio DRY (Don't Repeat Yourself) foi rigorosamente aplicado através da criação de componentes altamente reutilizáveis, garantindo que a lógica de apresentação de elementos repetitivos fosse escrita uma única vez:

O Componente StatBar.jsx: O card de Pokémon possui seis estatísticas base (HP, Attack, Defense, etc.). Em vez de escrever o HTML e o CSS da linha da barra de progresso seis vezes, o StatBar foi criado para abstrair essa visualização. O PokemonCard simplesmente mapeia a lista de estatísticas recebidas e renderiza uma instância de StatBar para cada uma, passando os valores como props.

3.2. Uso de Hooks e Otimização

O projeto utiliza Componentes Funcionais e Hooks do React para modernizar o código e gerenciar o estado de forma eficaz:

useState: Essencial para o gerenciamento de dados dinâmicos no PokedexContainer e para o estado local do input no SearchBar.

useEffect: Usado no PokedexContainer para gerenciar o ciclo de vida, especificamente para a chamada de dados assíncrona inicial (carregamento de um Pokémon aleatório ao montar o componente).

useCallback: Aplicado à função fetchPokemon para memorizar a função. Isso impede que ela seja recriada em cada re-render do PokedexContainer, otimizando a performance, especialmente quando a função é passada como prop para componentes filhos.

3.3. Rigor na Tipagem e Documentação

PropTypes: A biblioteca prop-types foi utilizada em todos os Componentes Presentational. Esta prática é crucial para documentar quais dados um componente espera receber e quais são os tipos (ex: string, number, func, object), garantindo a robustez do sistema e prevenindo erros causados por dados de tipo incorreto.

4. Aprendizados

O principal aprendizado reside na demonstração prática de como o Padrão Container/Presenter facilita a manutenibilidade e a escalabilidade do projeto. Ao isolar a lógica de dados no PokedexContainer, alcançamos benefícios claros:

- **Previsibilidade e Isolamento de Falhas:** Quaisquer alterações futuras na lógica de paginação, nos limites de busca da API (ex: mudar de 1025 para 1300 Pokémon) ou na fonte de dados, podem ser feitas exclusivamente no Container. Os componentes visuais (PokemonCard ou StatBar) permanecem imunes a essas mudanças, reduzindo drasticamente o risco de introduzir bugs visuais ao atualizar a lógica de negócios.

- **Organização Lógica:** O código está naturalmente organizado em pastas semânticas (containers e components). Isso facilita a onboarding de novos desenvolvedores e acelera a localização de funcionalidades: lógica está em containers, interface em components.

- **Aumento da Testabilidade:** Os Componentes Presentational, como o StatBar.jsx, tornaram-se puros e dependentes apenas de suas props. Isso significa que eles podem ser testados unitariamente de forma isolada, verificando apenas se a saída HTML está correta para uma determinada entrada de props, sem a necessidade de simular toda a infraestrutura da API.