Modelo de Projeto

Para um projeto de aprendizado, vamos criar um aplicativo de lista de tarefas simples. Esse projeto se beneficiará do uso de técnicas de Engenharia de Software, como o desenvolvimento de Test Driven Development (TDD), além de ter espaço para melhorias futuras. Vamos modelar o sistema conforme os requisitos.

Requisitos:

- O usuário pode adicionar uma nova tarefa.
- O usuário pode marcar uma tarefa como concluída.
- O usuário pode excluir uma tarefa.

Diagrama de Classes

Dado que este é um aplicativo simples, nosso Diagrama de Classes será relativamente direto. Vamos ter duas classes principais:

- **App**: Esta é a nossa classe principal que gerencia o estado do aplicativo e inclui métodos para adicionar, remover e atualizar tarefas.
- **App** tem dois estados:
    - **newTaskText** - um estado para armazenar o texto da nova tarefa.
    - **tasks** - um estado para armazenar a lista de tarefas.
- **App** possui três métodos principais:
    - **handleAddTask** - para adicionar uma nova tarefa à lista.
    - **handleToggleTaskDone** - para marcar uma tarefa como concluída ou não concluída.
    - **handleDeleteTask** - para excluir uma tarefa da lista.
- **App** também usa dois efeitos colaterais (com useEffect):
    - um para inicializar a lista de tarefas do localStorage quando o componente é montado.
    - outro para atualizar a lista de tarefas no localStorage sempre que a lista de tarefas muda.
- **Task:** Esta é uma classe para representar uma tarefa. Ela tem uma propriedade id para identificação, description para descrição da tarefa e *completed* para o estado de conclusão da tarefa.

Modelagem de Dados

A modelagem de dados está sendo gerida através do estado do componente do React. A estrutura de dados principal aqui é a matriz tasks de objetos Task, onde cada Task é um objeto com três propriedades:

- id: número único para identificar a tarefa.
- text: string para representar o texto da tarefa.
- done: booleano para indicar se a tarefa foi concluída

Erros e Melhorias

1. **Persistência dos dados**: A aplicação não persiste nos dados. Quando o componente é desmontado, todas as tarefas são perdidas. Uma solução seria armazenar as tarefas no localStorage ou usar um banco de dados.
2. **Manipulação de erros:** A aplicação não tem um tratamento de erros robusto. Por exemplo, o usuário pode adicionar uma tarefa vazia. Uma melhoria seria adicionar verificações para prevenir isso.
3. **Experiência do usuário:** A aplicação pode ser melhorada adicionando mais feedbacks visuais ao usuário. Por exemplo, poderia haver um indicador de carregamento quando uma tarefa está sendo adicionada ou removida.
4. **Estilos CSS:** Existem muitos estilos inline que poderiam ser extraídos para classes CSS separadas para melhor realização e manutenção.
5. **Testes:** Os testes atuais não cobrem todos os cenários possíveis. Poderiam ser adicionados mais testes para verificar o comportamento do aplicativo em diferentes casos de uso.

Testes realizados

```
import { fireEvent, render, screen } from "@testing-library/react"
import App from "./App";

test("Adicionar Tarefa", () => {
  render(<App />);
  const input = screen.getByPlaceholderText(/Nova Tarefa/i);
  fireEvent.change(input, { target: { value: "Levar o lixo" } });
  const addBtn = screen.getByText(/Adicionar Tarefa/i);
  fireEvent.click(addBtn);
  expect(screen.getByText("Levar o lixo")).toBeInTheDocument();
});

test("Finalizar Tarefa", () => {
  render(<App />);
  const input = screen.getByPlaceholderText(/Nova Tarefa/i);
  fireEvent.change(input, { target: { value: "Comprar pão" } });
  const addBtn = screen.getByText(/Adicionar Tarefa/i);
  fireEvent.click(addBtn);
  const finalizeBtn = screen.getByText(/Finalizar Tarefa/i);
  fireEvent.click(finalizeBtn);
  expect(screen.getByText(/Finalizada/i)).toBeInTheDocument();
});

test("Deletar Tarefa", () => {
  render(<App />);
  const input = screen.getByPlaceholderText(/Nova Tarefa/i);
  fireEvent.change(input, { target: { value: "Fazer jantar" } });
  const addBtn = screen.getByText(/Adicionar Tarefa/i);
  fireEvent.click(addBtn);
  const deleteBtn = screen.getByText(/Deletar/i);
  fireEvent.click(deleteBtn);
  expect(screen.queryByText("Fazer jantar")).toBeNull();
});
```

Solução dos problemas

1. Persistência dos dados

A aplicação atualmente armazena as tarefas em memória e elas são perdidas quando o componente é desmontado. Para garantir a persistência dos dados, é recomendado armazenar as tarefas em uma fonte de armazenamento, como o localStorage ou um banco de dados.

Uma abordagem simples seria utilizar o localStorage para armazenar as tarefas. No exemplo atual, já existe uma lógica para salvar as tarefas no localStorage sempre que houver uma alteração no estado tasks. No entanto, ao carregar a aplicação, as tarefas não são recuperadas do localStorage. Para corrigir isso, foi adicionado um

trecho de código no useEffect inicial para buscar as tarefas do localStorage e preencher o estado tasks com elas:

```
useEffect(() => {
  localStorage.setItem("tasks", JSON.stringify(tasks));
}, [tasks]);
```

Ao atualizar o status tasks em outras partes do código, as tarefas serão automaticamente salvas no localStorage. Isso garante que as tarefas persistam entre as sessões do usuário.

Para uma solução mais robusta e escalável, considere utilizar um banco de dados, como o MongoDB ou o Firebase Realtime Database, para armazenar as tarefas. Essas soluções permitem uma gestão mais avançada dos dados, como a sincronização em tempo real entre vários usuários.

2. Manipulação de erros

A aplicação atual não tem um tratamento robusto para manipulação de erros. Por exemplo, é possível adicionar uma tarefa vazia, o que não é desejado. Uma melhoria recomendada é adicionar verificações e validações para prevenir erros comuns.

No código atual, já existe uma verificação para garantir que a tarefa não seja vazia antes de adicioná-la. Caso contrário, exibe-se uma mensagem de erro para o usuário:

```
if (newTaskText.trim() === "") {
  message.error("A tarefa não pode ser vazia.");
  return;
}
```

Essa verificação impede que o usuário adicione uma tarefa vazia. No entanto, existem outros cenários de erros que podem ser considerados, como lidar com erros de rede ou erros ao recuperar as tarefas do armazenamento.

Para melhorar ainda mais a manipulação de erros, você pode utilizar técnicas como try-catch para capturar erros específicos e fornecer feedbacks mais informativos ao usuário. Além disso, considere adicionar validações adicionais, como limites de caracteres ou restrições específicas, dependendo dos requisitos do seu aplicativo.

3. Experiência do usuário

Para melhorar a experiência do usuário, é recomendado adicionar mais feedbacks visuais para fornecer informações sobre o estado atual da aplicação. Alguns exemplos de melhorias para a experiência do usuário incluem:

- Adicionar um indicador de carregamento durante a adição ou remoção de tarefas. Isso ajuda o usuário a entender que uma ação está em andamento.
- Utilizar animações suaves durante a adição, remoção ou atualização das tarefas para tornar as transições mais agradáveis aos olhos do usuário.
- Utilizar notificações ou mensagens para fornecer feedback sobre o resultado das ações realizadas, como sucesso na adição de uma tarefa ou erro na exclusão.

Essas melhorias podem ser alcançadas utilizando bibliotecas de animação, como React Transition Group ou Framer Motion, e componentes de notificação, como react-toastify ou antd's message, para fornecer um feedback visual mais rico para o usuário.

4. Estilos CCC

No código fornecido, existem muitos estilos CSS definidos inline. Uma melhoria recomendada é extrair esses estilos para classes CSS separadas ou componentes estilizados, o que facilitará a reutilização e a manutenção dos estilos.

Por exemplo, em vez de definir estilos diretamente nos elementos HTML, você pode criar classes CSS ou estilizar componentes utilizando bibliotecas como styled-components ou CSS Modules.

Exemplo:

```
// Em vez de:
<span style={{ fontSize: "18px" }}>{task.text}</span>

// Utilize uma classe CSS:
<span className="task-text">{task.text}</span>

// Ou utilize styled-components:
import styled from "styled-components";

const TaskText = styled.span`
  font-size: 18px;
`;

// E utilize o componente estilizado:
<TaskText>{task.text}</TaskText>
```