Documentação do Sistema

1. Arquitetura do Sistema

A aplicação é composta por um front-end desenvolvido em **React**, com gerenciamento de estado local, navegação baseada em rotas e integração com APIs externas para operações CRUD.

Estrutura Geral

- **Componentização**: Cada funcionalidade foi dividida em componentes reutilizáveis, organizados por finalidade:
  - `Main`: Página inicial que exibe os posts.
  - `Lista`: Página de listagem detalhada dos posts.
  - `Editar`: Formulário para atualizar as informações de posts.
  - `Login`: Página de autenticação.
  - `Header`: Responsável pelo menu de navegação e busca por categorias.
- **Roteamento**: Utiliza o `react-router-dom` para navegação:
  - Rotas públicas: Acessíveis sem autenticação (ex.: Login e Home).
  - Rotas privadas: Protegidas e acessíveis apenas para usuários autenticados (ex.: Professores, Lista, Editar).
- **Estilização**:
  - Combinação de **styled-components** para estilização local dos componentes.
  - Uso de CSS tradicional para estilos globais e componentes mais simples.

Principais Tecnologias

- **React**: Framework principal para desenvolvimento do front-end.
- **React Router Dom**: Biblioteca de roteamento.
- **Styled-Components**: Estilização de componentes de maneira modular e dinâmica.
- **Fetch API**: Utilizada para consumir endpoints REST.

2. Uso da Aplicação

Funcionalidades

- **Login**:
  - Permite que o usuário se autentique com credenciais (e-mail e senha).
  - Salva o token de autenticação no `localStorage` com expiração de 20 minutos.
  - Após login bem-sucedido, o usuário é redirecionado para a rota protegida.
- **Busca e Filtro**:

    - Campo de pesquisa no `Header` filtra os posts por categoria.
    - É exibido um conjunto único de resultados na página inicial.
- **Listagem de Postagens**:
    - Exibe todas as postagens disponíveis com informações detalhadas.
    - Oferece opções de edição e exclusão.
- **Edição de Postagens**:
    - Permite que o usuário busque uma postagem pelo ID e atualize seus dados.
    - Valida os campos do formulário antes de enviar as alterações para a API.
- **Rotas Protegidas**:
    - Rotas como Professores, Lista e Editar verificam se o token armazenado é válido antes de conceder acesso.

Como Usar

1. **Login**: O usuário precisa autenticar-se para acessar rotas protegidas.
2. **Busca**: Digite ao menos 3 caracteres na barra de pesquisa para filtrar resultados.
3. **Listar e Gerenciar Posts**:
    - Acesse a página de lista para visualizar os posts.
    - Utilize as opções de "Editar" ou "Deletar".
4. **Editar Postagem**:
    - Insira o ID da postagem na página de edição.
    - Altere os campos desejados e salve as alterações.

3. Relato de Experiências e Desafios

Pontos Fortes

- **Modularidade**: A estrutura baseada em componentes facilitou a manutenção e reutilização do código.
- **Responsividade**: O uso de styled-components permitiu criar estilos dinâmicos que se adaptam bem a diferentes tamanhos de tela.
- **Experiência do Usuário**: A integração com a API forneceu dados em tempo real, melhorando a usabilidade.

Desafios Enfrentados

1. **Gerenciamento de Estado**:
    - Garantir que a busca filtrasse apenas categorias únicas sem redundâncias foi um desafio inicial.
    - Solução: Uso de `Set` para manipular os resultados de maneira eficiente.
2. **Autenticação e Segurança**:
    - Manter o token de autenticação seguro e lidar com sua expiração exigiu planejamento.
    - Solução: Armazenar o token no `localStorage` com validação contínua.
3. **Edição Dinâmica**:

    - Implementar um formulário que atualizasse os campos com base no ID da postagem foi complexo.
    - Solução: Consumo de endpoints específicos da API e uso do estado local para carregar e enviar dados.
4. **Tratamento de Erros**:
    - Identificar mensagens de erro claras e apresentá-las ao usuário de forma amigável foi essencial.
    - Solução: Implementar mensagens de feedback em cada ação, como login, edição e exclusão.

4. Lições Aprendidas

- Dividir funcionalidades em pequenos componentes ajudou a manter o código limpo e fácil de compreender.
- Planejar bem as interações com a API foi crucial para evitar inconsistências e problemas de sincronização.