Para adaptar sua aplicação React para utilizar o **React Router**, você precisará seguir alguns passos para integrar a navegação baseada em rotas. Isso permitirá que você gerencie melhor as diferentes telas e componentes da sua aplicação, especialmente no contexto de autenticação de usuários.

Passo 1: Instalar o React Router

Primeiro, você precisa instalar o pacote `react-router-dom` em sua aplicação:

```
npm install react-router-dom
```

Passo 2: Configurar as Rotas no `App.js`

Você vai modificar o `App.js` para incluir o `BrowserRouter` e definir as rotas usando o `Switch` e o `Route` do React Router.

Atualização do `App.js`:

```
import React, { useContext } from 'react';
import { BrowserRouter as Router, Route, Switch, Redirect } from 'react-
router-dom';
import './styles.css';
import Header from './components/Header';
import { UsuarioProvider, UsuarioContext } from './context/UsuarioContext';
import SignIn from './components/SignIn';
import Quadros from './components/Quadros';

function App() {
  return (
    <UsuarioProvider>
      <Router>
        <Header />
        <Main />
      </Router>
    </UsuarioProvider>
  );
}

function Main() {
  const { usuario } = useContext(UsuarioContext);

  return (
    <Switch>
      {/* Rota de login */}
      <Route exact path="/login">
        {usuario ? <Redirect to="/quadros" /> : <SignIn />}
      </Route>

      {/* Rotas protegidas */}
      <PrivateRoute path="/quadros" component={Quadros} />
```

```
      {/* Redirecionar para /quadros se estiver logado, senão para /login
*/}
      <Route exact path="/">
        {usuario ? <Redirect to="/quadros" /> : <Redirect to="/login" />}
      </Route>

      {/* Rota para páginas não encontradas */}
      <Route path="*">
        <h1>Página não encontrada</h1>
      </Route>
    </Switch>
  );
}

export default App;
```

Explicação:

- **UsuarioProvider**: Envolvemos todo o aplicativo com o `UsuarioProvider` para que o contexto do usuário esteja disponível em toda a aplicação.
- **Router**: Envolvemos o aplicativo com o `Router` para habilitar o roteamento.
- **Main Component**: Criamos um componente `Main` que usa o `useContext` para acessar o `usuario` e define as rotas.
- **Rota de Login**:
  - Se o usuário **não está logado**, renderiza o componente `SignIn`.
  - Se o usuário **está logado**, redireciona para `/quadros`.
- **Rotas Protegidas**:
  - Usamos um componente `PrivateRoute` para proteger rotas que exigem autenticação.
- **Rota Padrão (`/`)**:
  - Se o usuário **está logado**, redireciona para `/quadros`.
  - Se o usuário **não está logado**, redireciona para `/login`.
- **Rota de Páginas Não Encontradas**:
  - Qualquer rota que não corresponda às anteriores renderiza uma mensagem de "Página não encontrada".

Passo 3: Criar o Componente `PrivateRoute`

Você precisará criar um componente para gerenciar rotas protegidas, redirecionando usuários não autenticados para a página de login.

Crie um arquivo `PrivateRoute.js` em `components`:

```
// components/PrivateRoute.js
import React, { useContext } from 'react';
import { Route, Redirect } from 'react-router-dom';
import { UsuarioContext } from '../context/UsuarioContext';

const PrivateRoute = ({ component: Component, ...rest }) => {
  const { usuario } = useContext(UsuarioContext);

  return (
    <Route
      {...rest}
      render={(props) =>
        usuario ? (
          <Component {...props} />
        ) : (
          <Redirect to={{ pathname: '/login', state: { from: props.location
} }} />
        )
      }
    />
  );
};

export default PrivateRoute;
```

Explicação:

- **PrivateRoute**: Verifica se o usuário está autenticado.
  - Se **estiver**, renderiza o componente solicitado.
  - Se **não estiver**, redireciona para `/login`, podendo armazenar a rota original em `state` para uso futuro.

Passo 4: Atualizar o `Header.js` para Usar Links do React Router

Substitua as tags `<a>` por `<Link>` do `react-router-dom` para evitar recarregamentos de página.

Atualização do `Header.js`:

```
import React from 'react';
import { Link } from 'react-router-dom';
import Logout from './Logout';
import { UsuarioContext } from '../context/UsuarioContext';

export default function Header() {
  const { usuario } = React.useContext(UsuarioContext);
  return (
    <header>
      <nav className="navbar">
        <div className="navbar-left">
```

```
          <Link to="/">
            <img src="logo.png" alt="Ghost Company" className="logo" />
          </Link>
        </div>
        <div className="navbar-center">
          <ul className="nav-links">
            <li>
              <Link to="/">Início</Link>
            </li>
            <li>
              <Link to="/perfil">Perfil</Link>
            </li>
            <li>
              <Link to="/sistemas">Sistemas</Link>
            </li>
          </ul>
        </div>
        <div className="navbar-right">
          {usuario ? (
            <div>
              <span>Olá, </span>
              <span>{usuario.nome} </span>
              {usuario.admin && <span className="dot"></span>}
               
              <Logout />
            </div>
          ) : (
            <span>Não autenticado</span>
          )}
        </div>
      </nav>
    </header>
  );
}
```

Explicação:

- **Uso do `<Link>`**: O componente `<Link>` substitui o `<a>` e evita recarregar a página ao navegar entre rotas.

Passo 5: Atualizar o Componente `Logout` para Redirecionar Após Logout

Após o logout, você pode redirecionar o usuário para a página de login.

Atualização do `Logout.js`:

```
import React, { useContext } from 'react';
import { useHistory } from 'react-router-dom';
import { UsuarioContext } from '../context/UsuarioContext';

const Logout = () => {
```

```
  const { setUsuario } = useContext(UsuarioContext);
  const history = useHistory();

  const handleLogout = () => {
    setUsuario(null); // Limpa as informações do usuário
    history.push('/login'); // Redireciona para a página de login
  };

  return <button onClick={handleLogout}>Sair</button>;
};

export default Logout;
```

Explicação:

- **useHistory**: Permite programaticamente navegar entre rotas.
- **history.push('/login')**: Redireciona o usuário para a página de login após o logout.

Passo 6: Ajustar o Componente `SignIn` para Redirecionar Após Login

Após um login bem-sucedido, redirecione o usuário para a rota protegida.

Atualização do `SignIn.js`:

```
import React, { useState, useContext, useEffect, useRef } from 'react';
import { useHistory } from 'react-router-dom';
import { UsuarioContext } from '../context/UsuarioContext';

const SignIn = () => {
  const emailInputRef = useRef(null);
  const { setUsuario } = useContext(UsuarioContext);
  const [email, setEmail] = useState('');
  const [senha, setSenha] = useState('');
  const [erro, setErro] = useState('');
  const history = useHistory();

  useEffect(() => {
    if (emailInputRef.current) {
      emailInputRef.current.focus();
    }
  }, []);

  const handleSubmit = (e) => {
    e.preventDefault();
    fetch('http://localhost:5000/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email, senha }),
    })
```

```
      .then((resposta) => {
        if (resposta.ok) {
          return resposta.json();
        } else {
          return resposta.json().then((erroDados) => {
            throw new Error(erroDados.message || 'Erro ao fazer login.');
          });
        }
      })
      .then((dados) => {
        const token = dados.token;
        setUsuario({ id: dados.id, nome: dados.nome, admin: dados.ehAdmin,
token: token });
        history.push('/quadros'); // Redireciona após login
      })
      .catch((erro) => {
        console.error('Erro na requisição:', erro);
        setErro(erro.message || 'Erro na conexão com o servidor.');
      });
  };

  return (
    <div className="signin">
      <h2>Entrar</h2>
      {erro && <p style={{ color: 'red' }}>{erro}</p>}
      <form onSubmit={handleSubmit}>
        <div>
          <label>Email:</label>
          <input
            type="text"
            value={email}
            ref={emailInputRef}
            onChange={(e) => setEmail(e.target.value)}
            required
          />
        </div>
        <div>
          <label>Senha:</label>
          <input
            type="password"
            value={senha}
            onChange={(e) => setSenha(e.target.value)}
            required
          />
        </div>
        <button type="submit">Entrar</button>
      </form>
    </div>
  );
};

export default SignIn;
```

Explicação:

- **useHistory**: Utilizado para redirecionar o usuário após o login bem-sucedido.
- **history.push('/quadros')**: Redireciona para a rota protegida `/quadros`.

Passo 7: Ajustar o Componente `Quadros` (Opcional)

Se você planeja ter múltiplas rotas dentro de `Quadros`, pode ajustar o componente para suportar sub-rotas.

Exemplo Simplificado:

Se não for necessário, você pode manter o `Quadros.js` como está. Caso contrário, você pode utilizar o `useParams` ou `useRouteMatch` do `react-router-dom` para gerenciar parâmetros de rota.

Passo 8: Atualizar Links de Navegação no `Header.js`

Certifique-se de que todos os links no `Header` estejam usando o componente `<Link>` e apontando para as rotas corretas.

Exemplo:

```
<li>
  <Link to="/quadros">Quadros</Link>
</li>
```

Passo 9: Testar a Aplicação

- Inicie sua aplicação com `npm start`.
- Navegue para `/login` e tente fazer login.
- Verifique se, após o login, você é redirecionado para `/quadros`.
- Teste o logout e verifique se é redirecionado para `/login`.

Considerações Finais

- **Persistência de Autenticação**: Você já está usando o `localStorage` para persistir o estado do usuário. Certifique-se de que isso está funcionando corretamente após as mudanças.
- **Proteção de Rotas**: O componente `PrivateRoute` garante que apenas usuários autenticados possam acessar certas rotas.
- **Redirecionamentos Apropriados**: A lógica de redirecionamento em `App.js` e nos componentes `SignIn` e `Logout` garante uma experiência de usuário fluida.
- **Atualização de Componentes**: Componentes como `Header` agora utilizam o React Router para navegação, evitando recarregamentos desnecessários da página.

Recursos Adicionais

- Documentação do React Router

- Guia de Autenticação com React Router

Espero que esta resposta o ajude a integrar o React Router em sua aplicação. Se tiver alguma dúvida ou enfrentar algum problema, sinta-se à vontade para perguntar!