1. Estrutura do Projeto

- **React com React Router**: Usaremos o React Router para gerenciar as rotas e limitar o acesso a funcionalidades específicas para cada tipo de usuário.
- **Autenticação e Autorização**: Para garantir que cada tipo de usuário tenha acesso apenas às suas funcionalidades, implementaremos uma autenticação simples, podendo evoluir para OAuth2 ou JWT.
- **Gestão de Consultas e Exames**: Criar componentes específicos para marcar e gerenciar consultas e exames.
- **Notificações**: Usar uma API para notificações ou gerenciamento de estado global (por exemplo, Redux ou Context API) para enviar notificações para pacientes e médicos.

2. Estrutura de Pastas

bash

Copiar código

```
/src
 /components
  /auth
   - Login.js
  /dashboard
   - AdminDashboard.js
   - DoctorDashboard.js
   - PatientDashboard.js
  /consultations
   - ConsultationForm.js
   - ConsultationList.js
  /exams
   - ExamSchedule.js
   - ExamResults.js
 /routes
  - PrivateRoute.js
  - AdminRoutes.js
  - DoctorRoutes.js
  - PatientRoutes.js
```

/services

- authService.js
- consultationService.js
- examService.js

- App.js
- index.js

3. Rotas e Autenticação

Implementar as rotas utilizando o React Router:

javascript

Copiar código

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Login from './components/auth/Login';
import AdminDashboard from './components/dashboard/AdminDashboard';
import DoctorDashboard from './components/dashboard/DoctorDashboard';
import PatientDashboard from './components/dashboard/PatientDashboard';
import PrivateRoute from './routes/PrivateRoute';

function App() {
 return (
  <Router>
   <Switch>
    <Route path="/login" component={Login} />
    <PrivateRoute path="/admin" component={AdminDashboard} role="admin" />
    <PrivateRoute path="/doctor" component={DoctorDashboard} role="doctor" />
    <PrivateRoute path="/patient" component={PatientDashboard} role="patient" />
   </Switch>
  </Router>
 );
}

export default App;
```

4. Autorização por Tipos de Usuário

A função PrivateRoute restringirá o acesso com base no tipo de usuário:

javascript

Copiar código

```
import { Route, Redirect } from 'react-router-dom';
import authService from '../services/authService';

const PrivateRoute = ({ component: Component, role, ...rest }) => {
 const user = authService.getCurrentUser();

 return (
  <Route
   {...rest}
   render={props =>
    user && user.role === role ? (
     <Component {...props} />
    ) : (
     <Redirect to="/login" />
    )
   }
  />
 );
};

export default PrivateRoute;
```

5. Serviço de Autenticação

authService.js será responsável por autenticar e armazenar informações sobre o usuário logado:

javascript

Copiar código

```
const authService = {
```

```
 login: (username, password) => {
  // Lógica de autenticação simulada ou chamada a uma API
  const users = [
   { username: 'admin', password: '123', role: 'admin' },
   { username: 'doctor', password: '123', role: 'doctor' },
   { username: 'patient', password: '123', role: 'patient' },
  ];
  const user = users.find(u => u.username === username && u.password === password);
  if (user) {
   localStorage.setItem('user', JSON.stringify(user));
  }
  return user;
 },
 getCurrentUser: () => JSON.parse(localStorage.getItem('user')),
 logout: () => localStorage.removeItem('user'),
};

export default authService;
```

6. Componentes Específicos para Cada Usuário

Criar dashboards diferentes para cada tipo de usuário.

AdminDashboard.js:

javascript

Copiar código

```
const AdminDashboard = () => (
 <div>
  <h2>Admin Dashboard</h2>
  <p>Gerenciar consultas, exames, e visualizar logs de usuários.</p>
 </div>
);
```

DoctorDashboard.js:

javascript

Copiar código

```
const DoctorDashboard = () => (
 <div>
  <h2>Dashboard Médico</h2>
  <p>Visualizar novos pacientes, gerenciar consultas e exames.</p>
 </div>
);
```

PatientDashboard.js:

javascript

Copiar código

```
const PatientDashboard = () => (
 <div>
  <h2>Dashboard Paciente</h2>
  <p>Visualizar consultas, exames e resultados.</p>
 </div>
);
```

7. Gestão de Consultas e Exames

Criar um serviço para interagir com as funcionalidades de consulta e exames. No consultationService.js e examService.js, implemente funções para simular a gestão de consultas e exames, que poderiam ser adaptadas para chamadas de API reais:

javascript

Copiar código

```
const consultationService = {
 getConsultations: () => [
  { id: 1, doctor: 'Dr. Smith', patient: 'John Doe', date: '2024-09-12' },
 ],
 scheduleConsultation: (consultation) => {
  // Adiciona a consulta ao sistema
 },
};


export default consultationService;
```

8. Notificações

A lógica de notificações pode ser implementada no estado global (via Context API) ou utilizando notificações push.

9. Leitura e Escrita de Arquivo .txt

O sistema pode exportar logs de atividades dos usuários em arquivos .txt, utilizando uma função para gerar e baixar os arquivos:

javascript

Copiar código

```
function generateTxtLog(userLogs) {
  const element = document.createElement("a");
  const file = new Blob([userLogs], { type: 'text/plain' });
  element.href = URL.createObjectURL(file);
  element.download = "userLogs.txt";
  document.body.appendChild(element);
  element.click();
}
```

Essa estrutura permite o desenvolvimento de um sistema hospitalar com funcionalidades específicas para cada usuário, garantindo segurança por meio de rotas privadas e controle de acesso.

Comece com o ChatG