

Aqui está a versão reescrita do seu prompt — mantive absolutamente todos os pontos e detalhes, apenas removi as menções ao **Base44**, de forma que o documento fique completo, limpo e sem perder nada importante:

---

# Prompt Detalhado: Prontuário Família - Seu Gestor de Saúde Familiar 360°

## Nome do Aplicativo

Prontuário Família

---

## Visão Geral

O **Prontuário Família** é uma plataforma web intuitiva e segura, projetada para ser a central de informações médicas de toda a sua família. Ele elimina a necessidade de papéis, planilhas ou diferentes aplicativos, consolidando o histórico de saúde, agendamentos e necessidades de cada membro em um único ambiente organizado.

Com foco na **usabilidade** e na **proteção de dados**, o aplicativo garante que você tenha acesso rápido e detalhado a informações cruciais a qualquer momento e em qualquer lugar.

---

## Público-Alvo

Pais, responsáveis, cuidadores ou qualquer pessoa que gerencie a saúde de múltiplos indivíduos e necessite de uma ferramenta centralizada para acompanhar, registrar e acessar de forma eficiente o histórico médico familiar.

---

## Funcionalidades em Detalhe

### 1. Dashboard (Tela Principal - "Prontuário Médico")

- **Visão Abrangente:** Exibe um resumo vital da saúde familiar.
- **Próximas Consultas:**
  - Lista as consultas agendadas (status "*scheduled*") em ordem cronológica.

- Detalhes: Nome do paciente, especialidade, médico, data e hora.
  - Localização: Exibe endereço da consulta (se informado).
  - **Integração Google Maps:** botão "Ver no Mapa" (se `location_url` preenchido).
  - **Integração Google Agenda:** botão "Add na Agenda", pré-preenchendo evento no Google Calendar (nome do paciente, tipo de consulta, médico, data, hora, duração padrão de 1h e local).
  - **Pendências (Avisos/Lembretes):**
    - Lista de "Pedidos Pendentes" (status *"pending"*) com paciente, título e data.
    - Anexos: link para visualizar fotos/PDFs anexados.
    - Gestão rápida: botão "Marcar como feito".
  - **Pacientes Cadastrados:**
    - Lista alfabética com cards contendo: foto, nome, idade, tipo sanguíneo, médico responsável, nº de alergias.
    - Clique no card → perfil detalhado.
    - Botão "Cadastrar Paciente".
  - **Últimas Atualizações:** log de atividades recentes de todo o app.
  - **Botão "Registro Rápido" (⚡):** acesso direto ao fluxo em 3 passos. (no topo da dashboard)
- 

## 2. Perfil Detalhado do Paciente (Tela "PatientDetails")

- **Navegação Rápida:** botões "Voltar" e "Editar Perfil".
- **Informações Pessoais:**
  - Foto de perfil (ou inicial do nome).
  - Nome, idade, tipo sanguíneo.
  - **Contato de Emergência:** nome e telefone (clícável em mobile).

- **Plano de Saúde:** nome, nº carteirinha, validade, médico responsável (nome, especialidade, contato).
- **Alergias:** lista.
- **Documentos Anexos:** carteirinha do plano (frente/verso) e identidade (frente/verso).
- **Abas de Informações Detalhadas:**
  - Design responsivo.
  - Cada aba → últimos 5 registros + botão "Ver Todos".
  - **Segurança (Senha de Dados Sensíveis):**
    - Opção configurável.
    - Se ativa → abas *Histórico*, *Incidentes*, *Pendências* e *Senhas* ficam bloqueadas.
    - Desbloqueio exige senha.
  - **Aba Exames:** lista com tipo, data, médico solicitante + ações (Editar, Apagar, Visualizar).
  - **Aba Medicações:** nome, dosagem, frequência, início + ações (Editar, Apagar, Visualizar).
  - **Aba Consultas:** especialidade, médico, data, hora, local + ações (Editar, Apagar, Visualizar).
  - **Aba Histórico:** doenças crônicas, cirurgias, internações, vacinas (Editar, Apagar, Visualizar).
  - **Aba Incidentes:** quedas, queimaduras, etc. + botão "Registrar Novo" (Editar, Apagar, Visualizar).
  - **Aba Pendências:** lista ativa/concluída + botões Adicionar, Marcar Feito/Pendente, Editar, Apagar.
  - **Aba Senhas (Credenciais):**
    - Nome do serviço, URL (clicável), usuário.
    - Senha oculta por padrão (•••••).

- Botões: Mostrar/ocultar, Copiar, Apagar.
- 

### 3. Cadastro/Edição de Paciente (Tela "PatientProfile")

- Formulário completo.
  - **Foto de Perfil:** upload.
  - **Documentos:** upload da carteirinha e identidade.
  - **Segurança e Privacidade:**
    - Switch "Proteger dados sensíveis com senha?".
    - Confirmação exigindo senha atual se tentar desativar.
    - Campo para cadastrar senha de acesso sensível.
- 

### 4. Registro Rápido (Modal/Overlay)

- **Acesso Simplificado:** pelo botão ⚡ no Dashboard.
  - **Fluxo em 3 Passos:**
    1. Selecionar Familiar (dropdown alfabético).
    2. Escolher Tipo (Exame, Medicação, Consulta, Histórico, Incidente, Pendência, Senha).
    3. Preencher Detalhes Mínimos:
      - Data (default = hoje).
      - Descrição (texto livre + sugestões rápidas).
      - Upload de anexos (PDF, PNG, JPG até 20MB, com progresso).
  - **Gravação Inteligente:** salva formatado no paciente correto.
  - **Feedback Visual:** mensagem clara após salvar.
-

## 5. Telas de Gestão Completas

- Páginas para cada tipo de registro: Exames, Medicações, Consultas, Histórico, Pedidos, Senhas.
  - Acessíveis pelo Dashboard ou perfil.
  - CRUD completo (Adicionar, Editar, Apagar).
  - Filtros e busca para facilitar.
- 

## 6. Navegação (Menu Lateral)

- Persistente.
  - Links para Dashboard e Busca.
  - Lista de pacientes (ordem alfabética, com atalhos).
  - Info do usuário logado (nome, e-mail) + botão Sair.
- 

## Tecnologia e Experiência do Usuário

- **Frontend Moderno:** React, Shadcn/UI (componentes), Lucide React (ícones) e Tailwind CSS (design limpo, moderno e responsivo).
  - **Responsividade:** funciona bem em desktop, tablet e celular.
  - **Autenticação e Armazenamento de Arquivos:** infraestrutura segura e escalável.
- 

## Valor e Benefícios

- **Organização Total:** centraliza todas as informações médicas da família.
- **Acesso Rápido e Seguro:** dados protegidos, acessíveis em segundos.
- **Decisão Informada:** histórico completo para compartilhar com profissionais de saúde.

- **Paz de Espírito:** nunca mais esquecer consultas, medicações ou pedidos médicos.

O **Prontuário Família** é o seu parceiro confiável na **gestão da saúde familiar**.

---

## Conteúdo do Aplicativo: Prontuário Saúde Família

```
/
├── entities/
│   ├── Patient.json
│   ├── MedicalHistory.json
│   ├── Exam.json
│   ├── Medication.json
│   ├── Appointment.json
│   ├── MedicalRequest.json
│   ├── Credential.json
│   ├── Incident.json
│   └── IncidentPhoto.json
├── pages/
│   ├── Dashboard.js
│   ├── PatientProfile.js
│   ├── Exams.js
│   ├── MedicalHistory.js
│   ├── Medications.js
│   ├── Appointments.js
│   ├── MedicalRequests.js
│   ├── Search.js
│   └── PatientDetails.js
├── components/
│   ├── exams/
│   │   ├── ExamUploadDialog.jsx
│   │   ├── ExamCard.jsx
│   │   └── ExamViewer.jsx
│   └── QuickRegister.jsx
└── layout.js
```

---

## Conteúdo dos Arquivos:

---

### entities/Patient.json

```
{
  "name": "Patient",
  "type": "object",
  "properties": {
    "full_name": {
      "type": "string",
      "description": "Nome completo do paciente"
    },
    "photo_url": {
      "type": "string",
      "description": "URL da foto de perfil do paciente"
    }
  }
}
```

```

    "birth_date": {
      "type": "string",
      "format": "date",
      "description": "Data de nascimento"
    },
    "blood_type": {
      "type": "string",
      "enum": ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"],
      "description": "Tipo sanguíneo"
    },
    "allergies": {
      "type": "array",
      "items": {"type": "string"},
      "description": "Lista de alergias"
    },
    "emergency_contact_name": {
      "type": "string",
      "description": "Nome do contato de emergência"
    },
    "emergency_contact_phone": {
      "type": "string",
      "description": "Telefone do contato de emergência"
    },
    "insurance_plan": {
      "type": "string",
      "description": "Plano de saúde/convênio"
    },
    "insurance_number": {
      "type": "string",
      "description": "Número do convênio"
    },
    "insurance_validity": {
      "type": "string",
      "format": "date",
      "description": "Validade do convênio"
    },
    "insurance_card_front_url": { "type": "string", "description": "URL da foto da frente da carteirinha" },
    "insurance_card_back_url": { "type": "string", "description": "URL da foto do verso da carteirinha" },
    "id_card_front_url": { "type": "string", "description": "URL da foto da frente da identidade" },
    "id_card_back_url": { "type": "string", "description": "URL da foto do verso da identidade" },
    "primary_doctor": {
      "type": "string",
      "description": "Médico responsável"
    },
    "primary_doctor_specialty": {
      "type": "string",
      "description": "Especialidade do médico responsável"
    },
    "primary_doctor_contact": {
      "type": "string",
      "description": "Contato do médico responsável"
    },
    "sensitive_data_password_active": { "type": "boolean", "default": false, "description": "Ativa a senha para dados sensíveis" },
    "sensitive_data_password": { "type": "string", "description": "Senha para acessar dados sensíveis" }
  },
  "required": ["full_name", "birth_date"]
}

```

---

## entities/MedicalHistory.json

```

{
  "name": "MedicalHistory",
  "type": "object",
  "properties": {
    "patient_id": { "type": "string", "description": "ID do paciente" },
    "type": { "type": "string", "enum": ["chronic_disease", "surgery", "hospitalization", "vaccination"], "description": "Tipo de registro médico" },

```

```
    "title": {"type": "string", "description": "Título/nome do procedimento/doença"},
    "description": {"type": "string", "description": "Descrição detalhada"},
    "date": {"type": "string", "format": "date", "description": "Data do evento"},
    "doctor": {"type": "string", "description": "Médico responsável"},
    "hospital": {"type": "string", "description": "Hospital/clínica"},
    "notes": {"type": "string", "description": "Observações adicionais"}
  },
  "required": ["patient_id", "type", "title", "date"]
}
```

---

## entities/Exam.json

```
{
  "name": "Exam",
  "type": "object",
  "properties": {
    "patient_id": {"type": "string", "description": "ID do paciente"},
    "exam_type": {"type": "string", "description": "Tipo de exame"},
    "exam_date": {"type": "string", "format": "date", "description": "Data do exame"},
    "requesting_doctor": {"type": "string", "description": "Médico solicitante"},
    "files": {"type": "array", "items": {"type": "string", "description": "URLs dos arquivos do exame"},
    "pdf_url": {"type": "string", "description": "URL do PDF consolidado"},
    "results_summary": {"type": "string", "description": "Resumo dos resultados"},
    "observations": {"type": "string", "description": "Observações sobre o exame"},
    "status": {"type": "string", "enum": ["pending", "completed", "reviewed"], "default": "completed", "description": "Status do exame"}
  },
  "required": ["patient_id", "exam_type", "exam_date"]
}
```

---

## entities/Medication.json

```
{
  "name": "Medication",
  "type": "object",
  "properties": {
    "patient_id": {"type": "string", "description": "ID do paciente"},
    "commercial_name": {"type": "string", "description": "Nome comercial do medicamento"},
    "generic_name": {"type": "string", "description": "Nome genérico"},
    "dosage": {"type": "string", "description": "Dosagem"},
    "frequency": {"type": "string", "description": "Frequência de uso"},
    "usage_type": {"type": "string", "enum": ["continuous", "temporary"], "default": "continuous", "description": "Tipo de uso (contínuo ou temporário)"},
    "time_of_day": {"type": "string", "enum": ["morning", "afternoon", "night", "dawn", "any"], "default": "morning", "description": "Período do dia para tomar"},
    "start_date": {"type": "string", "format": "date", "description": "Data de início"},
    "end_date": {"type": "string", "format": "date", "description": "Data de fim (se aplicável)"},
    "prescribing_doctor": {"type": "string", "description": "Médico que prescreveu"},
    "indication": {"type": "string", "description": "Indicação/motivo do uso"},
    "status": {"type": "string", "enum": ["active", "discontinued", "completed"], "default": "active", "description": "Status da medicação"},
    "side_effects": {"type": "string", "description": "Efeitos colaterais observados"}
  },
  "required": ["patient_id", "commercial_name", "dosage", "frequency", "start_date"]
}
```

---

## entities/Appointment.json

```
{
  "name": "Appointment",
  "type": "object",
  "properties": {
    "patient_id": {"type": "string", "description": "ID do paciente"},
  }
}
```



```

    "date": {"type": "string", "format": "date-time", "description": "Data e hora da consulta"},
    "doctor": {"type": "string", "description": "Nome do médico"},
    "specialty": {"type": "string", "description": "Especialidade médica"},
    "location": {"type": "string", "description": "Endereço do local da consulta"},
    "location_url": {"type": "string", "description": "URL do Google Maps para o local"},
    "type": {"type": "string", "enum": ["consultation", "follow_up", "routine", "emergency"], "description": "Tipo de consulta"},
    "status": {"type": "string", "enum": ["scheduled", "completed", "cancelled"], "default": "scheduled", "description": "Status da consulta"},
    "summary": {"type": "string", "description": "Resumo da consulta"},
    "recommendations": {"type": "string", "description": "Recomendações médicas"},
    "next_appointment": {"type": "string", "format": "date", "description": "Próxima consulta sugerida"}
  },
  "required": ["patient_id", "date", "doctor", "specialty"]
}

```

---

## entities/MedicalRequest.json

```

{
  "name": "MedicalRequest",
  "type": "object",
  "properties": {
    "patient_id": {
      "type": "string",
      "description": "ID do paciente"
    },
    "type": {
      "type": "string",
      "enum": ["exam_request", "referral", "prescription", "medical_certificate"],
      "description": "Tipo de pedido médico"
    },
    "title": {
      "type": "string",
      "description": "Título do pedido"
    },
    "description": {
      "type": "string",
      "description": "Descrição detalhada"
    },
    "requesting_doctor": {
      "type": "string",
      "description": "Médico solicitante"
    },
    "target_specialty": {
      "type": "string",
      "description": "Especialidade de destino (para encaminhamentos)"
    },
    "urgency": {
      "type": "string",
      "enum": ["routine", "urgent", "emergency"],
      "default": "routine",
      "description": "Nível de urgência"
    },
    "request_date": {
      "type": "string",
      "format": "date",
      "description": "Data do pedido"
    },
    "due_date": {
      "type": "string",
      "format": "date",
      "description": "Prazo para execução"
    },
    "status": {
      "type": "string",
      "enum": ["pending", "scheduled", "completed", "cancelled"],
      "default": "pending",
      "description": "Status do pedido"
    }
  }
}

```

```
    },
    "notes": {
      "type": "string",
      "description": "Observações adicionais"
    },
    "files": {
      "type": "array",
      "items": { "type": "string" },
      "description": "URLs dos arquivos do pedido"
    }
  },
  "required": ["patient_id", "type", "title", "requesting_doctor", "request_date"]
}
```

---

## entities/Credential.json

```
{
  "name": "Credential",
  "type": "object",
  "properties": {
    "patient_id": {"type": "string", "description": "ID do paciente ao qual esta credencial pertence"},
    "service_name": {"type": "string", "description": "Nome do serviço (ex: Laboratório Fleury, Portal Unimed)"},
    "url": {"type": "string", "format": "uri", "description": "Link para o portal do serviço"},
    "username": {"type": "string", "description": "Nome de usuário ou e-mail de acesso"},
    "password": {"type": "string", "description": "Senha de acesso"},
    "notes": {"type": "string", "description": "Observações adicionais (ex: 'usar CPF sem pontos')"}
  },
  "required": ["patient_id", "service_name"]
}
```

---

## entities/Incident.json

```
{
  "name": "Incident",
  "type": "object",
  "properties": {
    "patient_id": {"type": "string", "description": "ID do Paciente"},
    "title": {"type": "string", "description": "Título do incidente"},
    "description": {"type": "string", "description": "Descrição detalhada do incidente"},
    "start_date": {"type": "string", "format": "date", "description": "Data de início do incidente"},
    "status": {"type": "string", "enum": ["active", "resolved"], "default": "active", "description": "Status do incidente"}
  },
  "required": ["patient_id", "title", "start_date"]
}
```

---

## entities/IncidentPhoto.json

```
{
  "name": "IncidentPhoto",
  "type": "object",
  "properties": {
    "incident_id": {"type": "string", "description": "ID do Incidente"},
    "photo_url": {"type": "string", "description": "URL da foto"},
    "photo_date": {"type": "string", "format": "date", "description": "Data em que a foto foi tirada"},
    "notes": {"type": "string", "description": "Observações sobre a foto"}
  },
  "required": ["incident_id", "photo_url", "photo_date"]
}
```

---

## pages/Dashboard.js

```
import React, { useState, useEffect } from "react";
import { User } from "@entities/User";
import { Patient } from "@entities/Patient";
import { Appointment } from "@entities/Appointment";
import { MedicalRequest } from "@entities/MedicalRequest";
import { Exam } from "@entities/Exam";
import { Medication } from "@entities/Medication";
import { MedicalHistory } from "@entities/MedicalHistory";

import { Card, CardContent, CardHeader, CardTitle, CardFooter } from "@components/ui/card";
import { Button } from "@components/ui/button";
import { Link } from "react-router-dom";
import { createPageUrl } from "@utils";

import {
  Heart,
  Users,
  Plus,
  UserPlus,
  Activity,
  Calendar,
  ClipboardList,
  Clock,
  MapPin,
  ExternalLink,
  CheckCircle,
  Bell
} from "lucide-react";
import { Badge } from "@components/ui/badge";
import { format, formatDistanceToNow, addHours } from "date-fns";
import { ptBR } from "date-fns/locale";
import { Avatar, AvatarImage, AvatarFallback } from "@components/ui/avatar";

import QuickRegister from "../components/QuickRegister";

const RecentActivityLog = () => {
  const [activities, setActivities] = useState([]);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    const fetchActivities = async () => {
      try {
        const [patients, exams, medications, appointments, history, requests] = await Promise.all([
          Patient.list(),
          Exam.list('-updated_date', 10),
          Medication.list('-updated_date', 10),
          Appointment.list('-updated_date', 10),
          MedicalHistory.list('-updated_date', 10),
          MedicalRequest.list('-updated_date', 10),
        ]);
      }
    };

    const patientMap = new Map(patients.map(p => [p.id, p.full_name]));

    const getTitle = (item, type) => {
      switch(type) {
        case 'Exame': return item.exam_type;
        case 'Medicação': return item.commercial_name;
        case 'Consulta': return `${item.specialty} com Dr(a). ${item.doctor}`;
        case 'Histórico': return item.title;
        case 'Pedido Médico': return item.title;
        default: return 'Registro';
      }
    };

    const allActivities = [
      ...exams.map(i => ({ ...i, type: 'Exame', patientName: patientMap.get(i.patient_id), title: getTitle(i, 'Exame') })),
      ...medications.map(i => ({ ...i, type: 'Medicação', patientName: patientMap.get(i.patient_id), title: getTitle(i, 'Medicação') })),
      ...appointments.map(i => ({ ...i, type: 'Consulta', patientName: patientMap.get(i.patient_id), title: getTitle(i, 'Consulta') })),
    ];
  });
}
```

```

        ...history.map(i => ({ ...i, type: 'Histórico', patientName: patientMap.get(i.patient_id), title: getTitle(i, 'Histórico') })),
        ...requests.map(i => ({ ...i, type: 'Pedido Médico', patientName: patientMap.get(i.patient_id), title: getTitle(i, 'Pedido Médico') })),
    ];

    allActivities.sort((a, b) => new Date(b.updated_date) - new Date(a.updated_date));
    setActivities(allActivities.slice(0, 10));
  } catch (error) {
    console.error("Erro ao buscar atividades recentes:", error);
  } finally {
    setIsLoading(false);
  }
};

fetchActivities();
}, []);

if (isLoading) {
  return <p className="text-sm text-gray-500">Carregando atividades...</p>;
}

return (
  <div className="space-y-4">
    {activities.length > 0 ? activities.map(item => (
      <div key={` ${item.type}-${item.id}`} className="flex items-start gap-4">
        <div className="flex-shrink-0 w-8 h-8 rounded-full bg-gray-100 flex items-center justify-center">
          <Activity className="w-4 h-4 text-gray-500"/>
        </div>
        <div className="flex-1">
          <p className="text-sm text-gray-800">
            <span className="font-semibold">{item.created_by || 'Sistema'}</span> atualizou um(a) <span
              className="font-semibold">{item.type}</span> para <span className="font-semibold">{item.patientName || 'Paciente
              desconhecido'}</span>.
          </p>
          <p className="text-sm text-gray-600">{item.title}</p>
          <p className="text-xs text-gray-400 mt-1">
            {formatDistanceToNow(new Date(item.updated_date), { addSuffix: true, locale: ptBR })}
          </p>
        </div>
      </div>
    )) : (
      <p className="text-sm text-gray-500">Nenhuma atividade recente.</p>
    )}
  </div>
);
};

```

```

export default function Dashboard() {
  const [user, setUser] = useState(null);
  const [patients, setPatients] = useState([]);
  const [upcomingAppointments, setUpcomingAppointments] = useState([]);
  const [pendingRequests, setPendingRequests] = useState([]);
  const [patientMap, setPatientMap] = useState(new Map());
  const [isLoading, setIsLoading] = useState(true);
  const [showQuickRegister, setShowQuickRegister] = useState(false);
  const [successMessage, setSuccessMessage] = useState("");

  useEffect(() => {
    loadDashboardData();
  }, []);

  const loadDashboardData = async () => {
    setIsLoading(true); // Set loading true at the beginning
    try {
      const currentUser = await User.me();
      setUser(currentUser);

      const [allPatients, allAppointments, allRequests] = await Promise.all([
        Patient.list('-created_date'),
        Appointment.list(),

```

```

    MedicalRequest.list()
  });

  // Ordenar pacientes alfabeticamente pelo nome completo
  const sortedPatients = allPatients.sort((a, b) => {
    const nameA = (a.full_name || "").toLowerCase();
    const nameB = (b.full_name || "").toLowerCase();
    return nameA.localeCompare(nameB);
  });

  setPatients(sortedPatients);

  const pMap = new Map(sortedPatients.map(p => [p.id, p.full_name]));
  setPatientMap(pMap);

  const now = new Date();
  setUpcomingAppointments(
    allAppointments
      .filter(a => a.status === 'scheduled' && new Date(a.date) >= now)
      .sort((a,b) => new Date(a.date) - new Date(b.date))
  );

  setPendingRequests(allRequests.filter(r => r.status === 'pending'));

} catch (error) {
  console.error("Erro ao carregar dados do dashboard:", error);
} finally {
  setIsLoading(false);
}
};

const markRequestAsCompleted = async (requestId) => {
  try {
    await MedicalRequest.update(requestId, { status: 'completed' });
    loadDashboardData(); // Recarregar os dados para atualizar a UI
  } catch (error) {
    console.error("Erro ao atualizar pedido:", error);
  }
};

const generateGoogleCalendarLink = (appointment) => {
  const startTime = new Date(appointment.date);
  const endTime = addHours(startTime, 1); // Adiciona 1 hora por padrão

  const formatToGoogle = (date) => date.toISOString().replace(/-/|\.d{3}/g, "");

  const details = `Consulta de ${appointment.specialty} com Dr(a). ${appointment.doctor} para ${patientMap.get(appointment.patient_id)}.`;

  const link = new URL("https://calendar.google.com/calendar/render");
  link.searchParams.append('action', 'TEMPLATE');
  link.searchParams.append('text', `Consulta: ${appointment.specialty} - ${patientMap.get(appointment.patient_id)}`);
  link.searchParams.append('dates', `${formatToGoogle(startTime)}/${formatToGoogle(endTime)}`);
  link.searchParams.append('details', details);
  if (appointment.location) {
    link.searchParams.append('location', appointment.location);
  }

  return link.href;
};

const calculateAge = (birthDate) => {
  if (!birthDate) return "N/A";
  return Math.floor((new Date() - new Date(birthDate)) / (365.25 * 24 * 60 * 60 * 1000));
};

const handleQuickRegisterSuccess = (message) => {
  setSuccessMessage(message);
  setTimeout(() => setSuccessMessage(""), 4000);
  loadDashboardData(); // Recarrega os dados para mostrar o novo registro

```

```

};

if (isLoading) {
  return (
    <div className="min-h-screen flex items-center justify-center">
      <div className="flex items-center gap-2">
        <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600"></div>
        <span className="text-gray-600">Carregando...</span>
      </div>
    </div>
  );
}

return (
  <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">
    <div className="max-w-7xl mx-auto space-y-8">
      {/* Header */}
      <div className="flex flex-col md:flex-row justify-between items-start md:items-center gap-4">
        <div>
          <h1 className="text-3xl font-bold text-gray-900 mb-2">
            Prontuário Familiar 🏠
          </h1>
          <p className="text-gray-600">Gerencie a saúde de toda a família em um só lugar</p>
        </div>
        <div className="flex gap-3">
          <Button
            onClick={() => setShowQuickRegister(true)}
            className="bg-gradient-to-r from-purple-500 to-purple-600 hover:from-purple-600 hover:to-purple-700"
          >
            ⚡ Registro Rápido
          </Button>
          <Link to={createUrl("PatientProfile")}>
            <Button className="bg-gradient-to-r from-green-500 to-green-600 hover:from-green-600 hover:to-green-700">
              <UserPlus className="w-4 h-4 mr-2" />
              Cadastrar Paciente
            </Button>
          </Link>
        </div>
      </div>
      {/* Success Message */}
      {successMessage && (
        <div className="bg-green-100 border border-green-400 text-green-700 px-4 py-3 rounded relative">
          <CheckCircle className="w-4 h-4 inline mr-2" />
          {successMessage}
        </div>
      )}
      {/* Painel de Avisos */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <Card className="shadow-lg border-0">
          <CardHeader>
            <CardTitle className="flex items-center gap-2">
              <Calendar className="w-5 h-5 text-indigo-600" />
              Próximas Consultas
            </CardTitle>
          </CardHeader>
          <CardContent className="space-y-4">
            {upcomingAppointments.length > 0 ? upcomingAppointments.slice(0, 3).map(appt => (
              <div key={appt.id} className="p-4 bg-indigo-50 rounded-lg hover:bg-indigo-100 transition-colors">
                <p className="font-semibold text-indigo-800">{patientMap.get(appt.patient_id)}</p>
                <p className="text-sm">{appt.specialty} com Dr(a). {appt.doctor}</p>
                <p className="text-sm font-medium text-blue-600 my-1">
                  {format(new Date(appt.date), "dd/MM/yyyy 'às' HH:mm", { locale: ptBR })}
                </p>
                {appt.location && (
                  <div className="flex items-center text-sm text-gray-600 gap-1">
                    <MapPin className="w-3 h-3"/> {appt.location}
                  </div>
                )}
              </div>
            )}
          </CardContent>
        </Card>
      </div>
    </div>
  </div>
);

```

```

<div className="flex gap-2 mt-2">
  {appt.location_url && (
    <a href__={appt.location_url} target="_blank" rel="noopener noreferrer">
      <Button size="sm" variant="outline">Ver no Mapa <ExternalLink className="w-3 h-3 ml-1"/></Button>
    </a>
  )}
  <a href__={generateGoogleCalendarLink(appt)} target="_blank" rel="noopener noreferrer">
    <Button size="sm" variant="outline">Add na Agenda <Calendar className="w-3 h-3 ml-1"/></Button>
  </a>
</div>
</div>
)) : (
  <p className="text-gray-500 text-center py-4">Nenhuma consulta agendada.</p>
)
</CardContent>
</Card>

```

```

<Card className="shadow-lg border-0">
  <CardHeader>
    <CardTitle className="flex items-center gap-2">
      <Bell className="w-5 h-5 text-orange-600" />
      Pendências
    </CardTitle>
  </CardHeader>
  <CardContent className="space-y-3">
    {pendingRequests.length > 0 ? pendingRequests.slice(0, 3).map(req => (
      <div key={req.id} className="p-3 bg-orange-50 rounded-lg hover:bg-orange-100 transition-colors">
        <div className="flex justify-between items-start">
          <div>
            <p className="font-semibold text-orange-800">{patientMap.get(req.patient_id)}</p>
            <p className="text-sm">{req.title}</p>
            <p className="text-xs text-gray-500">
              Pedido em {format(new Date(req.request_date), "dd/MM/yyyy", { locale: ptBR })}
            </p>
            {req.files && req.files.length > 0 &&
              <a href__={req.files[0]} target="_blank" rel="noopener noreferrer">
                <Button size="sm" variant="link" className="p-0 h-auto text-blue-600">Ver anexo</Button>
              </a>
            }
          </div>
          <Button size="sm" variant="ghost" className="text-green-600 hover:text-green-700" onClick={() =>
            markRequestAsCompleted(req.id)}>
            <CheckCircle className="w-4 h-4 mr-1"/> Marcar como feito
          </Button>
        </div>
      </div>
    ) : (
      <p className="text-gray-500 text-center py-4">Nenhuma pendência encontrada.</p>
    )
  </CardContent>
</Card>
</div>

```

```

{/* Lista de Pacientes */}
<Card className="shadow-lg border-0">
  <CardHeader>
    <div className="flex justify-between items-center">
      <CardTitle className="text-xl font-bold flex items-center gap-2">
        <Users className="w-6 h-6 text-blue-600" />
        Pacientes Cadastrados
      </CardTitle>
      <Link to={createPageUrl("PatientProfile")}>
        <Button variant="outline" className="hover:bg-blue-50">
          <Plus className="w-4 h-4 mr-2" />
          Novo Paciente
        </Button>
      </Link>
    </div>
  </CardHeader>

```

```

<CardContent>
{patients.length > 0 ? (
  <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
    {patients.map((patient) => (
      <Link key={patient.id} to={createPageUrl('PatientDetails?id=${patient.id}')}>
        <Card className="group hover:shadow-lg transition-all duration-300 cursor-pointer border hover:border-blue-200">
          <CardContent className="p-6">
            <div className="text-center space-y-3">
              <Avatar className="w-20 h-20 mx-auto border-4 border-white shadow-md">
                <AvatarImage src={patient.photo_url} alt={patient.full_name} />
                <AvatarFallback className="bg-gradient-to-r from-blue-500 to-blue-600 text-white text-2xl font-bold">
                  {patient.full_name?.charAt(0)?.toUpperCase() || 'P'}
                </AvatarFallback>
              </Avatar>
              <div>
                <h3 className="font-bold text-gray-900 group-hover:text-blue-600 transition-colors">
                  {patient.full_name}
                </h3>
                <div className="flex justify-center items-center gap-2 mt-2">
                  <Badge variant="outline">
                    {calculateAge(patient.birth_date)} anos
                  </Badge>
                  {patient.blood_type && (
                    <Badge variant="outline" className="bg-red-50 text-red-700 border-red-200">
                      {patient.blood_type}
                    </Badge>
                  )}
                </div>
              </div>
            </div>
          </CardContent>
          {patient.primary_doctor && (
            <p className="text-sm text-gray-500">
              Dr(a). {patient.primary_doctor}
            </p>
          )}
          {patient.allergies && patient.allergies.length > 0 && (
            <div className="flex flex-wrap gap-1 justify-center">
              <Badge variant="destructive" className="text-xs">
                {patient.allergies.length} alergia{patient.allergies.length > 1 ? 's' : ''}
              </Badge>
            </div>
          )}
          <p className="text-xs text-gray-400">
            Cadastrado em {format(new Date(patient.created_date), "dd/MM/yyyy", { locale: ptBR })}
          </p>
        </div>
      </CardContent>
    </Card>
  </Link>
    )}
  </div>
) : (
  <div className="text-center py-12">
    <Users className="w-16 h-16 text-gray-400 mx-auto mb-4" />
    <h3 className="text-lg font-semibold text-gray-900 mb-2">
      Nenhum paciente cadastrado
    </h3>
    <p className="text-gray-500 mb-4">
      Comece cadastrando o primeiro membro da família
    </p>
    <Link to={createPageUrl("PatientProfile")}>
      <Button className="bg-gradient-to-r from-green-500 to-green-600 hover:from-green-600 hover:to-green-700">
        <UserPlus className="w-4 h-4 mr-2" />
        Cadastrar Primeiro Paciente
      </Button>
    </Link>
  </div>
)
)

```



```

    </CardContent>
  </Card>

  { /* Últimas Atualizações */
  <Card className="shadow-lg border-0">
    <CardHeader>
      <CardTitle className="text-xl font-bold flex items-center gap-2">
        <Clock className="w-6 h-6 text-gray-600" />
        Últimas Atualizações no Prontuário
      </CardTitle>
    </CardHeader>
    <CardContent>
      <RecentActivityLog />
    </CardContent>
  </Card>

  { /* Quick Register Modal */
  <QuickRegister
    isOpen={showQuickRegister}
    onClose={() => setShowQuickRegister(false)}
    onSuccess={handleQuickRegisterSuccess}
  />
  </div>
</div>
);
}

```

---

## pages/PatientProfile.js

```

import React, { useState, useEffect, useRef } from "react";
import { User } from "@entities/User";
import { Patient } from "@entities/Patient";
import { UploadFile } from "@integrations/Core";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { Input } from "@components/ui/input";
import { Label } from "@components/ui/label";
import { Button } from "@components/ui/button";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@components/ui/select";
import { Badge } from "@components/ui/badge";
import { UserIcon, Save, Plus, X, Heart, Phone, Shield, ArrowLeft, Camera, Lock, Unlock } from "lucide-react";
import { Alert, AlertDescription, AlertTitle } from "@components/ui/alert";
import { useNavigate } from "react-router-dom";
import { createPageUrl } from "@utils";
import { Avatar, AvatarImage, AvatarFallback } from "@components/ui/avatar";
import { Switch } from "@components/ui/switch";
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogFooter, DialogClose } from "@components/ui/dialog";

export default function PatientProfile() {
  const navigate = useNavigate();
  const urlParams = new URLSearchParams(window.location.search);
  const patientId = urlParams.get('id');

  const fileInputRef = useRef(null);

  const [user, setUser] = useState(null);
  const [patient, setPatient] = useState(null);
  const [formData, setFormData] = useState({
    full_name: "",
    photo_url: "",
    birth_date: "",
    blood_type: "",
    allergies: [],
    emergency_contact_name: "",
    emergency_contact_phone: "",
    insurance_plan: "",
    insurance_number: "",
  });

```

```

insurance_validity: "",
insurance_card_front_url: "",
insurance_card_back_url: "",
id_card_front_url: "",
id_card_back_url: "",
primary_doctor: "",
primary_doctor_specialty: "",
primary_doctor_contact: "",
sensitive_data_password_active: false,
sensitive_data_password: ""
});
const [newAllergy, setNewAllergy] = useState("");
const [isLoading, setIsLoading] = useState(true);
const [isSaving, setIsSaving] = useState(false);
const [message, setMessage] = useState("");
const [imagePreview, setImagePreview] = useState("");

// State for password confirmation dialog
const [isConfirmingDisable, setIsConfirmingDisable] = useState(false);
const [passwordToConfirm, setPasswordToConfirm] = useState("");
const [confirmationError, setConfirmationError] = useState("");

useEffect(() => {
  loadPatientData();
}, [patientId]);

const loadPatientData = async () => {
  try {
    const currentUser = await User.me();
    setUser(currentUser);

    if (patientId) {
      // Carregando um paciente específico para edição
      const foundPatient = await Patient.filter({id: patientId});
      if (foundPatient.length > 0) {
        setPatient(foundPatient[0]);
        setFormData(foundPatient[0]);
        if(foundPatient[0].photo_url) {
          setImagePreview(foundPatient[0].photo_url);
        }
      }
    }
  } catch (error) {
    console.error("Erro ao carregar dados do paciente:", error);
  } finally {
    setIsLoading(false);
  }
};

const handleInputChange = (field, value) => {
  setFormData(prev => ({
    ...prev,
    [field]: value
  }));
};

const handleFileUpload = async (e, fieldName) => {
  const file = e.target.files[0];
  if (file) {
    setIsSaving(true);
    try {
      const { file_url } = await UploadFile({ file });
      handleInputChange(fieldName, file_url);
      setMessage(`Documento ${fieldName} carregado!`);
    } catch (error) {
      console.error(`Erro no upload do ${fieldName}:`, error);
      setMessage(`Erro ao fazer upload do documento.`);
    } finally {
      setIsSaving(false);
    }
  }
}

```

```

    }
  };

const handlePhotoUpload = async (e) => {
  const file = e.target.files[0];
  if (file) {
    setImagePreview(URL.createObjectURL(file));
    setIsSaving(true);
    try {
      const { file_url } = await UploadFile({ file });
      handleInputChange('photo_url', file_url);
      setMessage("Foto atualizada com sucesso!");
    } catch (error) {
      console.error("Erro no upload da foto:", error);
      setMessage("Erro ao fazer upload da foto.");
    } finally {
      setIsSaving(false);
    }
  }
};

const addAllergy = () => {
  if (newAllergy.trim()) {
    setFormData(prev => ({
      ...prev,
      allergies: [...(prev.allergies || []), newAllergy.trim()]
    }));
    setNewAllergy("");
  }
};

const removeAllergy = (index) => {
  setFormData(prev => ({
    ...prev,
    allergies: prev.allergies.filter((_, i) => i !== index)
  }));
};

const handlePasswordToggle = (checked) => {
  // If user is trying to DISABLE the password, show confirmation dialog
  // This dialog is only shown if the patient exists and had an active password
  if (!checked && patient && patient.sensitive_data_password_active) {
    setIsConfirmingDisable(true);
    setConfirmationError(""); // Clear previous errors
    setPasswordToConfirm(""); // Clear previous input
  } else {
    // Otherwise, just update the state as usual
    handleInputChange('sensitive_data_password_active', checked);
    // If activating, clear password field so user can set a new one
    if (checked && !formData.sensitive_data_password) {
      handleInputChange('sensitive_data_password', "");
    }
  }
};

const handleConfirmDisablePassword = () => {
  if (passwordToConfirm === patient.sensitive_data_password) {
    // Correct password, proceed with disabling
    handleInputChange('sensitive_data_password_active', false);
    // Optional: Clear the password field when disabling
    // handleInputChange('sensitive_data_password', "");
    setIsConfirmingDisable(false);
    setPasswordToConfirm("");
    setConfirmationError("");
  } else {
    // Incorrect password
    setConfirmationError("Senha incorreta. A desativação foi cancelada.");
  }
};

```

```

const handleSubmit = async (e) => {
  e.preventDefault();
  setIsSaving(true);

  try {
    if (patient) {
      await Patient.update(patient.id, formData);
      setMessage("Perfil atualizado com sucesso!");
    } else {
      const newPatient = await Patient.create(formData);
      setPatient(newPatient);
      setMessage("Paciente cadastrado com sucesso!");
    }
  }

  setTimeout(() => {
    setMessage("");
    navigate(createPageUrl("Dashboard"));
  }, 2000);
} catch (error) {
  console.error("Erro ao salvar perfil:", error);
  setMessage("Erro ao salvar perfil. Tente novamente.");
} finally {
  setIsSaving(false);
}
};

if (isLoading) {
  return (
    <div className="min-h-screen flex items-center justify-center">
      <div className="flex items-center gap-2">
        <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600"></div>
        <span className="text-gray-600">Carregando perfil...</span>
      </div>
    </div>
  );
}

return (
  <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">
    <Dialog open={isConfirmingDisable} onOpenChange={setIsConfirmingDisable}>
      <DialogContent>
        <DialogHeader>
          <DialogTitle>Confirmar Desativação da Senha</DialogTitle>
        </DialogHeader>
        <div className="space-y-4 py-4">
          <p>Para desativar a proteção por senha para dados sensíveis, por favor, insira a senha atual do paciente.</p>
          <Input
            type="password"
            placeholder="Digite a senha atual"
            value={passwordToConfirm}
            onChange={(e) => setPasswordToConfirm(e.target.value)}
            onKeyPress={(e) => e.key === 'Enter' && handleConfirmDisablePassword()}
          />
          {confirmationError && (
            <Alert variant="destructive" className="mt-2">
              <AlertDescription>{confirmationError}</AlertDescription>
            </Alert>
          )}
        </div>
        <DialogFooter>
          <DialogClose asChild>
            <Button variant="outline" onClick={() => setConfirmationError("")}>Cancelar</Button>
          </DialogClose>
          <Button onClick={handleConfirmDisablePassword}>Confirmar e Desativar</Button>
        </DialogFooter>
      </DialogContent>
    </Dialog>
  </div>

```

```

<div className="max-w-4xl mx-auto">
  <div className="mb-8 flex items-center gap-4">
    <Button
      variant="outline"
      onClick={() => navigate(createPageUrl("Dashboard"))}
    >
      <ArrowLeft className="w-4 h-4 mr-2" />
      Voltar
    </Button>
    <div>
      <h1 className="text-3xl font-bold text-gray-900 mb-2">
        {patient ? `Editando: ${patient.full_name}` : 'Cadastrar Novo Paciente'}
      </h1>
      <p className="text-gray-600">
        {patient ? 'Atualize as informações do paciente' : 'Adicione um novo membro da família'}
      </p>
    </div>
  </div>

  {message && (
    <Alert className="mb-6 border-green-200 bg-green-50">
      <AlertDescription className="text-green-800">{message}</AlertDescription>
    </Alert>
  )}

  <form onSubmit={handleSubmit} className="space-y-6">
    {/* Informações Pessoais */}
    <Card className="shadow-lg border-0">
      <CardHeader className="bg-gradient-to-r from-blue-500 to-blue-600 text-white rounded-t-lg">
        <CardTitle className="flex items-center gap-2">
          <UserIcon className="w-5 h-5" />
          Informações Pessoais
        </CardTitle>
      </CardHeader>
      <CardContent className="p-6 space-y-4">
        <div className="flex items-center gap-6">
          <Avatar className="w-24 h-24 border-4 border-white shadow-lg">
            <AvatarImage src={imagePreview} />
            <AvatarFallback className="bg-gray-200">
              <UserIcon className="w-12 h-12 text-gray-500" />
            </AvatarFallback>
          </Avatar>
          <div className="flex-1">
            <Label>Foto de Perfil</Label>
            <Input
              type="file"
              className="hidden"
              ref__={fileInputRef}
              onChange={handlePhotoUpload}
              accept="image/*"
            />
            <Button type="button" variant="outline" onClick={() => fileInputRef.current.click()}>
              <Camera className="w-4 h-4 mr-2" />
              Trocar Foto
            </Button>
            <p className="text-xs text-gray-500 mt-2">Clique para selecionar uma nova foto.</p>
          </div>
        </div>
      </CardContent>
    </div>
    <div className="grid md:grid-cols-2 gap-4">
      <div>
        <Label htmlFor="full_name">Nome Completo *</Label>
        <Input
          id="full_name"
          value={formData.full_name}
          onChange={(e) => handleInputChange('full_name', e.target.value)}
          placeholder="Nome completo do paciente"
          required
        />
      </div>
    </div>
  </div>

```

```

    <Label htmlFor="birth_date">Data de Nascimento *</Label>
    <Input
      id="birth_date"
      type="date"
      value={formData.birth_date}
      onChange={(e) => handleInputChange('birth_date', e.target.value)}
      required
    />
  </div>
</div>
<div>
  <Label htmlFor="blood_type">Tipo Sanguíneo</Label>
  <Select value={formData.blood_type} onChange={(value) => handleInputChange('blood_type', value)}>
    <SelectTrigger>
      <SelectValue placeholder="Selecione o tipo sanguíneo" />
    </SelectTrigger>
    <SelectContent>
      <SelectItem value="A+">A+</SelectItem>
      <SelectItem value="A-">A-</SelectItem>
      <SelectItem value="B+">B+</SelectItem>
      <SelectItem value="B-">B-</SelectItem>
      <SelectItem value="AB+">AB+</SelectItem>
      <SelectItem value="AB-">AB-</SelectItem>
      <SelectItem value="O+">O+</SelectItem>
      <SelectItem value="O-">O-</SelectItem>
    </SelectContent>
  </Select>
</div>
</CardContent>
</Card>

```

```

{/* Alergias */}
<Card className="shadow-lg border-0">
  <CardHeader className="bg-gradient-to-r from-red-500 to-red-600 text-white rounded-t-lg">
    <CardTitle className="flex items-center gap-2">
      <Shield className="w-5 h-5" />
      Alergias
    </CardTitle>
  </CardHeader>
  <CardContent className="p-6 space-y-4">
    <div className="flex gap-2">
      <Input
        value={newAllergy}
        onChange={(e) => setNewAllergy(e.target.value)}
        placeholder="Digite uma alergia"
        onKeyDown={(e) => e.key === 'Enter' && (e.preventDefault(), addAllergy())}
      />
      <Button type="button" onClick={addAllergy} variant="outline">
        <Plus className="w-4 h-4" />
      </Button>
    </div>
    <div className="flex flex-wrap gap-2">
      {formData.allergies?.map((allergy, index) => (
        <Badge key={index} variant="destructive" className="flex items-center gap-1">
          {allergy}
          <X
            className="w-3 h-3 cursor-pointer hover:bg-red-700 rounded-full p-0.5"
            onClick={() => removeAllergy(index)}
          />
        </Badge>
      ))}
    </div>
    {(!formData.allergies || formData.allergies.length === 0) && (
      <p className="text-gray-500 text-sm">Nenhuma alergia registrada</p>
    )}
  </CardContent>
</Card>

```

```

{/* Contato de Emergência */}
<Card className="shadow-lg border-0">

```

```

<CardHeader className="bg-gradient-to-r from-orange-500 to-orange-600 text-white rounded-t-lg">
  <CardTitle className="flex items-center gap-2">
    <Phone className="w-5 h-5" />
    Contato de Emergência
  </CardTitle>
</CardHeader>
<CardContent className="p-6 space-y-4">
  <div className="grid md:grid-cols-2 gap-4">
    <div>
      <Label htmlFor="emergency_contact_name">Nome do Contato</Label>
      <Input
        id="emergency_contact_name"
        value={formData.emergency_contact_name}
        onChange={(e) => handleInputChange('emergency_contact_name', e.target.value)}
        placeholder="Nome da pessoa de contato"
      />
    </div>
    <div>
      <Label htmlFor="emergency_contact_phone">Telefone</Label>
      <Input
        id="emergency_contact_phone"
        value={formData.emergency_contact_phone}
        onChange={(e) => handleInputChange('emergency_contact_phone', e.target.value)}
        placeholder="(11) 99999-9999"
      />
    </div>
  </div>
</CardContent>
</Card>

```

```

{/* Plano de Saúde */}
<Card className="shadow-lg border-0">
  <CardHeader className="bg-gradient-to-r from-green-500 to-green-600 text-white rounded-t-lg">
    <CardTitle className="flex items-center gap-2">
      <Heart className="w-5 h-5" />
      Plano de Saúde
    </CardTitle>
  </CardHeader>
  <CardContent className="p-6 space-y-4">
    <div className="grid md:grid-cols-2 gap-4">
      <div>
        <Label htmlFor="insurance_plan">Nome do Plano</Label>
        <Input
          id="insurance_plan"
          value={formData.insurance_plan}
          onChange={(e) => handleInputChange('insurance_plan', e.target.value)}
          placeholder="Ex: Unimed, Bradesco Saúde"
        />
      </div>
      <div>
        <Label htmlFor="insurance_number">Número do Cartão</Label>
        <Input
          id="insurance_number"
          value={formData.insurance_number}
          onChange={(e) => handleInputChange('insurance_number', e.target.value)}
          placeholder="Número do cartão do plano"
        />
      </div>
    </div>
    <div>
      <Label htmlFor="insurance_validity">Validade</Label>
      <Input
        id="insurance_validity"
        type="date"
        value={formData.insurance_validity}
        onChange={(e) => handleInputChange('insurance_validity', e.target.value)}
      />
    </div>
  </div>
  <div className="grid md:grid-cols-2 gap-4">
    <div>

```

```

        <Label>Carteirinha (Frente)</Label>
        <Input type="file" onChange={(e) => handleFileUpload(e, 'insurance_card_front_url')} accept="image/*,.pdf" />
        {formData.insurance_card_front_url && <a href__={formData.insurance_card_front_url} target="_blank"
className="text-blue-600 text-sm">Ver arquivo</a>}
      </div>
      <div>
        <Label>Carteirinha (Verso)</Label>
        <Input type="file" onChange={(e) => handleFileUpload(e, 'insurance_card_back_url')} accept="image/*,.pdf" />
        {formData.insurance_card_back_url && <a href__={formData.insurance_card_back_url} target="_blank"
className="text-blue-600 text-sm">Ver arquivo</a>}
      </div>
    </div>
    <div className="grid md:grid-cols-2 gap-4">
      <div>
        <Label>Identidade (Frente)</Label>
        <Input type="file" onChange={(e) => handleFileUpload(e, 'id_card_front_url')} accept="image/*,.pdf" />
        {formData.id_card_front_url && <a href__={formData.id_card_front_url} target="_blank" className="text-blue-600
text-sm">Ver arquivo</a>}
      </div>
      <div>
        <Label>Identidade (Verso)</Label>
        <Input type="file" onChange={(e) => handleFileUpload(e, 'id_card_back_url')} accept="image/*,.pdf" />
        {formData.id_card_back_url && <a href__={formData.id_card_back_url} target="_blank" className="text-blue-600
text-sm">Ver arquivo</a>}
      </div>
    </div>
  </CardContent>
</Card>

{ /* Médico Responsável */ }
<Card className="shadow-lg border-0">
  <CardHeader className="bg-gradient-to-r from-purple-500 to-purple-600 text-white rounded-t-lg">
    <CardTitle className="flex items-center gap-2">
      <UserIcon className="w-5 h-5" />
      Médico Responsável
    </CardTitle>
  </CardHeader>
  <CardContent className="p-6 space-y-4">
    <div className="grid md:grid-cols-2 gap-4">
      <div>
        <Label htmlFor="primary_doctor">Nome do Médico</Label>
        <Input
          id="primary_doctor"
          value={formData.primary_doctor}
          onChange={(e) => handleInputChange('primary_doctor', e.target.value)}
          placeholder="Dr. João Silva"
        />
      </div>
      <div>
        <Label htmlFor="primary_doctor_specialty">Especialidade</Label>
        <Input
          id="primary_doctor_specialty"
          value={formData.primary_doctor_specialty}
          onChange={(e) => handleInputChange('primary_doctor_specialty', e.target.value)}
          placeholder="Ex: Clínico Generalista"
        />
      </div>
    </div>
    <div>
      <Label htmlFor="primary_doctor_contact">Contato</Label>
      <Input
        id="primary_doctor_contact"
        value={formData.primary_doctor_contact}
        onChange={(e) => handleInputChange('primary_doctor_contact', e.target.value)}
        placeholder="Telefone ou e-mail"
      />
    </div>
  </CardContent>
</Card>

```



```

    {/* Segurança */}
    <Card className="shadow-lg border-0">
      <CardHeader className="bg-gradient-to-r from-gray-500 to-gray-600 text-white rounded-t-lg">
        <CardTitle className="flex items-center gap-2">
          <Lock className="w-5 h-5" />
          Segurança e Privacidade
        </CardTitle>
      </CardHeader>
      <CardContent className="p-6 space-y-4">
        <div className="flex items-center justify-between p-3 bg-gray-50 rounded-lg">
          <Label htmlFor="sensitive-switch" className="flex flex-col">
            <span className="font-medium">Proteger dados sensíveis com senha?</span>
            <span className="text-xs text-gray-500">Exigirá uma senha para ver exames, histórico, etc.</span>
          </Label>
          <Switch
            id="sensitive-switch"
            checked={formData.sensitive_data_password_active}
            onChange={handlePasswordToggle}
          />
        </div>
        {formData.sensitive_data_password_active && (
          <div>
            <Label htmlFor="sensitive_password">Senha de Acesso Sensível</Label>
            <Input
              id="sensitive_password"
              type="password"
              value={formData.sensitive_data_password}
              onChange={(e) => handleInputChange('sensitive_data_password', e.target.value)}
              placeholder={patient?.sensitive_data_password ? 'Deixe em branco para manter a senha atual' : 'Crie uma senha
simples'}
            />
            <p className="text-xs text-gray-500 mt-1">Use uma senha fácil de lembrar. Esta senha será solicitada para ver dados
sensíveis.</p>
          </div>
        )}
      </CardContent>
    </Card>

    <div className="flex justify-end gap-3">
      <Button
        type="button"
        variant="outline"
        onClick={() => navigate(createPageUrl("Dashboard"))}
      >
        Cancelar
      </Button>
      <Button
        type="submit"
        disabled={isSaving}
        className="bg-gradient-to-r from-blue-500 to-blue-600 hover:from-blue-600 hover:to-blue-700 px-8 py-3"
      >
        {isSaving ? (
          <>
            <div className="animate-spin rounded-full h-4 w-4 border-b-2 border-white mr-2"></div>
            Salvando...
          </>
        ) : (
          <>
            <Save className="w-4 h-4 mr-2" />
            {patient ? 'Atualizar Paciente' : 'Cadastrar Paciente'}
          </>
        )}
      </Button>
    </div>
  </form>
</div>
</div>
);
}

```

## pages/Exams.js

```
import React, { useState, useEffect } from "react";
import { Patient } from "@entities/Patient";
import { Exam } from "@entities/Exam";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { Link } from "react-router-dom";
import { createPageUrl } from "@utils";
import {
  TestTube,
  Search,
  Plus,
  ArrowLeft,
  FileText,
  Calendar,
  Users
} from "lucide-react";
import { Alert, AlertDescription } from "@components/ui/alert";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@components/ui/select";

import ExamUploadDialog from "../components/exams/ExamUploadDialog";
import ExamCard from "../components/exams/ExamCard";
import ExamViewer from "../components/exams/ExamViewer";

export default function Exams() {
  const urlParams = new URLSearchParams(window.location.search);
  const patientIdFromUrl = urlParams.get("patient");

  const [allPatients, setAllPatients] = useState([]);
  const [selectedPatient, setSelectedPatient] = useState(null);

  const [exams, setExams] = useState([]);
  const [filteredExams, setFilteredExams] = useState([]);
  const [searchTerm, setSearchTerm] = useState("");
  const [showUploadDialog, setShowUploadDialog] = useState(false);
  const [selectedExam, setSelectedExam] = useState(null);
  const [isLoading, setIsLoading] = useState(true);
  const [editingExam, setEditingExam] = useState(null);

  // Load initial data (all patients) on component mount
  useEffect(() => {
    loadInitialData();
  }, []);

  // Load exams when a patient is selected or changes
  useEffect(() => {
    if(selectedPatient) {
      loadExamsForPatient(selectedPatient.id);
    } else {
      setExams([]); // Clear exams if no patient is selected
    }
  }, [selectedPatient]);

  // Filter exams when search term or exams list changes
  useEffect(() => {
    filterExams();
  }, [searchTerm, exams]);

  const loadInitialData = async () => {
    setIsLoading(true);
    try {
      const patients = await Patient.list();
      setAllPatients(patients);
      if (patientIdFromUrl) {

```

```

    const patient = patients.find(p => p.id === patientIdFromUrl);
    if(patient) setSelectedPatient(patient);
    else if (patients.length > 0) { // If URL patient not found, default to first patient
      setSelectedPatient(patients[0]);
    }
  } else if (patients.length > 0) {
    setSelectedPatient(patients[0]); // Select the first patient if no ID in URL
  } else {
    setSelectedPatient(null); // No patients found at all
  }
} catch(e) {
  console.error("Erro ao carregar pacientes:", e);
  setSelectedPatient(null); // Ensure state is consistent on error
} finally {
  setIsLoading(false);
}
};

const loadExamsForPatient = async (patientId) => {
  setIsLoading(true);
  try {
    const patientExams = await Exam.filter({ patient_id: patientId }, '-exam_date');
    setExams(patientExams);
  } catch (error) {
    console.error("Erro ao carregar exames:", error);
    setExams([]);
  } finally {
    setIsLoading(false);
  }
};

const filterExams = () => {
  if (!searchTerm) {
    setFilteredExams(exams);
  } else {
    const filtered = exams.filter(exam =>
      exam.exam_type.toLowerCase().includes(searchTerm.toLowerCase()) ||
      exam.requesting_doctor?.toLowerCase().includes(searchTerm.toLowerCase()) ||
      exam.observations?.toLowerCase().includes(searchTerm.toLowerCase())
    );
    setFilteredExams(filtered);
  }
};

const handleSaveSuccess = (savedExam) => {
  // This now handles both creation and update
  loadExamsForPatient(selectedPatient.id);
  setShowUploadDialog(false);
  setEditingExam(null);
};

const handleEdit = (exam) => {
  setEditingExam(exam);
  setShowUploadDialog(true);
};

const handleDelete = async (examId) => {
  if (window.confirm("Tem certeza que deseja apagar este exame? Esta ação não pode ser desfeita.")) {
    try {
      await Exam.delete(examId);
      loadExamsForPatient(selectedPatient.id); // Refresh the list
    } catch (error) {
      console.error("Erro ao apagar exame:", error);
      // You could add a user-facing error message here
    }
  }
};

const handlePatientChange = (patientId) => {
  const patient = allPatients.find(p => p.id === patientId);

```

```

    setSelectedPatient(patient);
  };

  if (isLoading && (!selectedPatient || exams.length === 0)) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="flex items-center gap-2">
          <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600"></div>
          <span className="text-gray-600">
            {selectedPatient ? "Carregando exames..." : "Carregando pacientes..."}
          </span>
        </div>
      </div>
    );
  }

  // After loading, if there are no patients at all
  if (!isLoading && allPatients.length === 0) {
    return (
      <div className="min-h-screen flex flex-col items-center justify-center p-6 text-center">
        <Users className="w-16 h-16 text-gray-400 mx-auto mb-4" />
        <h2 className="text-2xl font-bold text-gray-800 mb-2">Nenhum Paciente Cadastrado</h2>
        <p className="text-gray-600 mb-6">Por favor, cadastre um paciente para começar a adicionar exames.</p>
        <Link to={createPageUrl("Dashboard")}>
          <Button>
            <ArrowLeft className="w-4 h-4 mr-2" />
            Voltar ao Dashboard
          </Button>
        </Link>
      </div>
    );
  }

  return (
    <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">
      <div className="max-w-7xl mx-auto space-y-6">
        {/* Header */}
        <div className="flex flex-col md:flex-row justify-between items-start md:items-center gap-4">
          <div>
            <h1 className="text-3xl font-bold text-gray-900 mb-2">
              Exames {selectedPatient ? `de ${selectedPatient.full_name}` : `da Família`}
            </h1>
            <p className="text-gray-600">Organize e visualize os exames de forma prática</p>
          </div>
          <div className="flex items-center gap-4">
            <Select onChange={handlePatientChange} value={selectedPatient?.id || ""}>
              <SelectTrigger className="w-[280px]">
                <SelectValue placeholder="Selecione um paciente..." />
              </SelectTrigger>
              <SelectContent>
                {allPatients.map(p => <SelectItem key={p.id} value={p.id}>{p.full_name}</SelectItem>)}
              </SelectContent>
            </Select>
            <Button
              onClick={() => { setEditingExam(null); setShowUploadDialog(true); }}
              disabled={!selectedPatient}
              className="bg-gradient-to-r from-blue-500 to-blue-600 hover:from-blue-600 hover:to-blue-700"
            >
              <Plus className="w-4 h-4 mr-2" />
              Novo Exame
            </Button>
          </div>
        </div>
      </div>

      {/* Search Bar */}
      <Card className="shadow-lg border-0">
        <CardContent className="p-4">
          <div className="relative">
            <Search className="absolute left-3 top-1/2 transform -translate-y-1/2 text-gray-400 w-5 h-5" />
            <Input

```

```

        placeholder="Pesquisar por tipo de exame, médico ou observações..."
        value={searchTerm}
        onChange={(e) => setSearchTerm(e.target.value)}
        className="pl-10 pr-4 py-2 w-full"
      />
    </div>
  </CardContent>
</Card>

{/* Stats */}
<div className="grid grid-cols-1 md:grid-cols-3 gap-4">
  <Card className="shadow-md border-0">
    <CardContent className="p-4">
      <div className="flex items-center gap-3">
        <div className="p-2 bg-blue-100 rounded-lg">
          <TestTube className="w-5 h-5 text-blue-600" />
        </div>
        <div>
          <p className="text-sm text-gray-500">Total de Exames</p>
          <p className="text-xl font-bold text-gray-900">{exams.length}</p>
        </div>
      </div>
    </CardContent>
  </Card>

  <Card className="shadow-md border-0">
    <CardContent className="p-4">
      <div className="flex items-center gap-3">
        <div className="p-2 bg-green-100 rounded-lg">
          <Calendar className="w-5 h-5 text-green-600" />
        </div>
        <div>
          <p className="text-sm text-gray-500">Este Mês</p>
          <p className="text-xl font-bold text-gray-900">
            {exams.filter(e => new Date(e.exam_date).getMonth() === new Date().getMonth()).length}
          </p>
        </div>
      </div>
    </CardContent>
  </Card>

  <Card className="shadow-md border-0">
    <CardContent className="p-4">
      <div className="flex items-center gap-3">
        <div className="p-2 bg-orange-100 rounded-lg">
          <FileText className="w-5 h-5 text-orange-600" />
        </div>
        <div>
          <p className="text-sm text-gray-500">Tipos Diferentes</p>
          <p className="text-xl font-bold text-gray-900">
            {new Set(exams.map(e => e.exam_type)).size}
          </p>
        </div>
      </div>
    </CardContent>
  </Card>
</div>

{/* Exams Grid */}
{filteredExams.length > 0 ? (
  <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
    {filteredExams.map((exam) => (
      <ExamCard
        key={exam.id}
        exam={exam}
        onView={setSelectedExam}
        onEdit={handleEdit}
        onDelete={handleDelete}
      />
    ))}
  </div>
)}

```

```

    </div>
  ) : (
    <Card className="shadow-lg border-0">
      <CardContent className="p-12 text-center">
        <TestTube className="w-12 h-12 text-gray-400 mx-auto mb-4" />
        <h3 className="text-lg font-semibold text-gray-900 mb-2">
          {searchTerm ? "Nenhum exame encontrado" : "Nenhum exame cadastrado"}
        </h3>
        <p className="text-gray-500 mb-4">
          {searchTerm
            ? "Tente alterar os termos de busca"
            : "Comece adicionando seu primeiro exame médico"
          }
        </p>
        {!searchTerm && (
          <Button
            onClick={() => { setEditingExam(null); setShowUploadDialog(true); }}
            className="bg-gradient-to-r from-blue-500 to-blue-600 hover:from-blue-600 hover:to-blue-700"
          >
            <Plus className="w-4 h-4 mr-2" />
            Adicionar Primeiro Exame
          </Button>
        )}
      </CardContent>
    </Card>
  )}

  {/* Upload Dialog */}
  <ExamUploadDialog
    isOpen={showUploadDialog}
    onClose={() => { setShowUploadDialog(false); setEditingExam(null); }}
    patientId={selectedPatient?.id}
    onSaveSuccess={handleSaveSuccess}
    examToEdit={editingExam}
  />

  {/* Exam Viewer */}
  <ExamViewer
    exam={selectedExam}
    onClose={() => setSelectedExam(null)}
  />
</div>
</div>
);
}

```

## pages/MedicalHistory.js

```

import React, { useState, useEffect } from "react";
import { User } from "@entities/User";
import { Patient } from "@entities/Patient";
import { MedicalHistory as MedicalHistoryEntity } from "@entities/MedicalHistory";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { Button } from "@components/ui/button";
import { Alert, AlertDescription } from "@components/ui/alert";
import { Plus, FileText, HeartPulse, Stethoscope, Syringe, Hospital } from "lucide-react";
import { format } from "date-fns";
import { ptBR } from "date-fns/locale";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@components/ui/select";

```

```

// Placeholder for a form component to be created
const HistoryForm = ({ isOpen, onClose, patientId, onHistoryCreated, historyItem }) => {
  const [formData, setFormData] = useState(historyItem || {
    type: "chronic_disease",
    title: "",
    description: "",
  });

```

```

    date: "",
    doctor: "",
    hospital: "",
    notes: ""
  });
  const [isSaving, setIsSaving] = useState(false);

  useEffect(() => {
    if(historyItem) {
      setFormData({
        ...historyItem,
        date: historyItem.date ? format(new Date(historyItem.date), 'yyyy-MM-dd') : ""
      });
    } else {
      setFormData({
        type: "chronic_disease",
        title: "",
        description: "",
        date: "",
        doctor: "",
        hospital: "",
        notes: ""
      });
    }
  }, [historyItem]);

  if (!isOpen) return null;

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSaving(true);
    try {
      const dataToSave = { ...formData, patient_id: patientId };
      if (historyItem) {
        await MedicalHistoryEntity.update(historyItem.id, dataToSave);
      } else {
        await MedicalHistoryEntity.create(dataToSave);
      }
      onHistoryCreated();
      onClose();
    } catch (error) {
      console.error("Failed to save history:", error);
    } finally {
      setIsSaving(false);
    }
  };

  return (
    <div className="fixed inset-0 bg-black/60 z-50 flex items-center justify-center p-4">
      <Card className="w-full max-w-lg">
        <CardHeader>
          <CardTitle>{historyItem ? 'Editar' : 'Adicionar'} Registro Histórico</CardTitle>
        </CardHeader>
        <CardContent>
          <form onSubmit={handleSubmit} className="space-y-4">
            {/* Form fields here, simplified for brevity */}
            <div>
              <label>Tipo</label>
              <select value={formData.type} onChange={e => setFormData({...formData, type: e.target.value})} className="w-full p-2 border rounded">
                <option value="chronic_disease">Doença Crônica</option>
                <option value="surgery">Cirurgia</option>
                <option value="hospitalization">Internação</option>
                <option value="vaccination">Vacina</option>
              </select>
            </div>
            <div>
              <label>Título</label>
              <input type="text" value={formData.title} onChange={e => setFormData({...formData, title: e.target.value})} required
                className="w-full p-2 border rounded" />

```

```

        </div>
        <div>
            <label>Data</label>
            <input type="date" value={formData.date} onChange={e => setFormData({...formData, date: e.target.value})} required
className="w-full p-2 border rounded" />
        </div>
        <div>
            <label>Descrição</label>
            <textarea value={formData.description} onChange={e => setFormData({...formData, description: e.target.value})}
className="w-full p-2 border rounded" />
        </div>
        <div className="flex justify-end gap-2">
            <Button type="button" variant="outline" onClick={onClose}>Cancelar</Button>
            <Button type="submit" disabled={isSaving}>{isSaving ? "Salvando..." : "Salvar"}</Button>
        </div>
    </form>
</CardContent>
</Card>
</div>
);
};

```

```

export default function MedicalHistoryPage() {
    const urlParams = new URLSearchParams(window.location.search);
    const patientIdFromUrl = urlParams.get('patient');

    const [allPatients, setAllPatients] = useState([]);
    const [selectedPatient, setSelectedPatient] = useState(null);
    const [history, setHistory] = useState([]);
    const [isLoading, setIsLoading] = useState(true);
    const [showForm, setShowForm] = useState(false);

    const historyTypes = {
        chronic_disease: { label: "Doenças Crônicas", icon: HeartPulse, color: "text-red-500" },
        surgery: { label: "Cirurgias", icon: Stethoscope, color: "text-blue-500" },
        hospitalization: { label: "Internações", icon: Hospital, color: "text-purple-500" },
        vaccination: { label: "Vacinas", icon: Syringe, color: "text-green-500" },
    };

    useEffect(() => {
        loadInitialData();
    }, []);

    useEffect(() => {
        if(selectedPatient) {
            loadHistoryForPatient(selectedPatient.id);
        } else {
            setHistory([]);
        }
    }, [selectedPatient]);

    const loadInitialData = async () => {
        setIsLoading(true);
        try {
            const patients = await Patient.list();
            setAllPatients(patients);
            if (patientIdFromUrl) {
                const patient = patients.find(p => p.id === patientIdFromUrl);
                if(patient) setSelectedPatient(patient);
                else if (patients.length > 0) { // Fallback if URL patient not found
                    setSelectedPatient(patients[0]);
                }
            } else if (patients.length > 0) {
                setSelectedPatient(patients[0]);
            }
        } catch(e) {
            console.error(e);
        } finally {
            setIsLoading(false);
        }
    }
}

```



```

    }

    const loadHistoryForPatient = async (patientId) => {
      setIsLoading(true);
      try {
        const historyData = await MedicalHistoryEntity.filter({ patient_id: patientId }, '-date');
        setHistory(historyData);
      } catch (error) {
        console.error("Erro ao carregar histórico:", error);
        setHistory([]);
      } finally {
        setIsLoading(false);
      }
    };

    const handleHistoryCreated = () => {
      if(selectedPatient) loadHistoryForPatient(selectedPatient.id);
    }

    const handlePatientChange = (patientId) => {
      const patient = allPatients.find(p => p.id === patientId);
      setSelectedPatient(patient);
    }

    if (isLoading && history.length === 0) return <p className="p-6">Carregando...</p>;

    return (
      <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">
        <div className="max-w-7xl mx-auto space-y-6">
          <div className="flex flex-col md:flex-row justify-between items-center gap-4">
            <div>
              <h1 className="text-3xl font-bold text-gray-900">Histórico Médico de {selectedPatient?.full_name}</h1>
              <p className="text-gray-600">Todos os seus eventos de saúde em um só lugar.</p>
            </div>
            <div className="flex items-center gap-4">
              <Select onChange={handlePatientChange} value={selectedPatient?.id || ""}>
                <SelectTrigger className="w-[280px]">
                  <SelectValue placeholder="Selecione um paciente..." />
                </SelectTrigger>
                <SelectContent>
                  {allPatients.map(p => <SelectItem key={p.id} value={p.id}>{p.full_name}</SelectItem>)}
                </SelectContent>
              </Select>
              <Button onClick={() => setShowForm(true)} disabled={!selectedPatient} className="bg-gradient-to-r from-blue-500 to-blue-600">
                <Plus className="w-4 h-4 mr-2" />
                Adicionar Registro
              </Button>
            </div>
          </div>
          <div>
            {!selectedPatient && <Alert className="mt-6"><AlertDescription>Selecione um paciente para visualizar o histórico.</AlertDescription></Alert>}

            {selectedPatient && (
              <div className="space-y-8">
                {Object.entries(historyTypes).map(([type, { label, icon: Icon, color }]) => {
                  const items = history.filter(h => h.type === type);
                  return (
                    <Card key={type} className="shadow-lg border-0">
                      <CardHeader>
                        <CardTitle className={`flex items-center gap-3 ${color}`}>
                          <Icon className="w-6 h-6" />
                          {label}
                        </CardTitle>
                      </CardHeader>
                      <CardContent>
                        {items.length > 0 ? (
                          <div className="space-y-4">
                            {items.map(item => (

```

```

        <div key={item.id} className="p-4 bg-gray-50 rounded-lg">
          <div className="flex justify-between items-start">
            <div>
              <h4 className="font-semibold text-gray-800">{item.title}</h4>
              <p className="text-sm text-gray-500">
                {format(new Date(item.date), "dd/MM/yyyy", { locale: ptBR })}
              </p>
            </div>
            {item.doctor && <span className="text-sm text-gray-600">Dr(a). {item.doctor}</span>}
          </div>
          {item.description && <p className="mt-2 text-gray-700">{item.description}</p>}
        </div>
      </div>
    </div>
  </div>
  <div>
    <p className="text-gray-500">Nenhum registro encontrado.</p>
  </div>
</CardContent>
</Card>
);
}}}
</div>
)}
<HistoryForm isOpen={showForm} onClose={() => setShowForm(false)} patientId={selectedPatient?.id}
onHistoryCreated={handleHistoryCreated} />
</div>
</div>
);
}

```

## pages/Medications.js

```

import React, { useState, useEffect } from "react";
import { Patient } from "@entities/Patient";
import { Medication as MedicationEntity } from "@entities/Medication";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { Button } from "@components/ui/button";
import { Alert, AlertDescription } from "@components/ui/alert";
import { Plus, Pill, Archive, Edit, Trash2 } from "lucide-react";
import { format } from "date-fns";
import { ptBR } from "date-fns/locale";
import { Badge } from "@components/ui/badge";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@components/ui/select";
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogTrigger, DialogFooter } from "@components/ui/dialog";
import { Input } from "@components/ui/input";
import { Label } from "@components/ui/label";
import { Textarea } from "@components/ui/textarea";

const MedicationForm = ({ isOpen, onClose, patientId, onMedicationCreated, medicationToEdit }) => {
  const [formData, setFormData] = useState({
    commercial_name: "",
    dosage: "",
    frequency: "",
    start_date: "",
    status: "active", // Default status
    usage_type: "continuous", // New field
    time_of_day: "morning", // New field
    generic_name: "", // New field
    end_date: "", // New field
    prescribing_doctor: "", // New field
    indication: "", // New field
    side_effects: "" // New field
  });
  const [isSaving, setIsSaving] = useState(false);

  useEffect(() => {

```

```

    if (medicationToEdit) {
      setFormData({
        ...medicationToEdit,
        start_date: medicationToEdit.start_date ? format(new Date(medicationToEdit.start_date), 'yyyy-MM-dd') : "",
        end_date: medicationToEdit.end_date ? format(new Date(medicationToEdit.end_date), 'yyyy-MM-dd') : "",
        // Ensure default values for new fields if they are null/undefined from backend
        usage_type: medicationToEdit.usage_type || "continuous",
        time_of_day: medicationToEdit.time_of_day || "morning",
        generic_name: medicationToEdit.generic_name || "",
        prescribing_doctor: medicationToEdit.prescribing_doctor || "",
        indication: medicationToEdit.indication || "",
        side_effects: medicationToEdit.side_effects || ""
      });
    } else {
      // Reset form for new medication
      setFormData({
        commercial_name: "", dosage: "", frequency: "", start_date: "", status: "active",
        usage_type: "continuous", time_of_day: "morning", generic_name: "", end_date: "",
        prescribing_doctor: "", indication: "", side_effects: ""
      });
    }
  }, [medicationToEdit]);

  if (!isOpen) return null;

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSaving(true);
    try {
      const dataToSave = {
        ...formData,
        patient_id: patientId,
        // Convert empty string dates to null for backend
        start_date: formData.start_date || null,
        end_date: formData.end_date || null,
      };

      if (medicationToEdit) {
        await MedicationEntity.update(medicationToEdit.id, dataToSave);
      } else {
        await MedicationEntity.create(dataToSave);
      }
      onMedicationCreated();
      onClose();
    } catch (error) {
      console.error("Failed to save medication:", error);
    } finally {
      setIsSaving(false);
    }
  };

  return (
    <DialogContent className="max-h-[90vh] overflow-y-auto">
      <DialogHeader><DialogTitle>{medicationToEdit ? "Editar" : "Adicionar"} Medicação</DialogTitle></DialogHeader>
      <form onSubmit={handleSubmit} className="space-y-4">
        <div className="grid grid-cols-2 gap-4">
          <div>
            <Label htmlFor="commercial_name">Nome Comercial *</Label>
            <Input id="commercial_name" value={formData.commercial_name} onChange={e => setFormData({...formData,
commercial_name: e.target.value})} required />
          </div>
          <div>
            <Label htmlFor="generic_name">Nome Genérico</Label>
            <Input id="generic_name" value={formData.generic_name} onChange={e => setFormData({...formData, generic_name:
e.target.value})} />
          </div>
        </div>
        <div className="grid grid-cols-2 gap-4">
          <div>
            <Label htmlFor="dosage">Dosagem *</Label>

```

```

        <Input id="dosage" value={formData.dosage} onChange={e => setFormData({...formData, dosage: e.target.value})}
required />
    </div>
    <div>
        <Label htmlFor="frequency">Frequência/Horários *</Label>
        <Input id="frequency" value={formData.frequency} onChange={e => setFormData({...formData, frequency:
e.target.value})} placeholder="Ex: 1x ao dia às 8h" required />
    </div>
</div>
<div>
    <div className="grid grid-cols-2 gap-4">
        <div>
            <Label htmlFor="usage_type">Tipo de Uso</Label>
            <Select value={formData.usage_type} onValueChange={v => setFormData({...formData, usage_type: v})}>
                <SelectTrigger id="usage_type">
                    <SelectValue placeholder="Selecione o tipo de uso"/>
                </SelectTrigger>
                <SelectContent>
                    <SelectItem value="continuous">Uso Contínuo</SelectItem>
                    <SelectItem value="temporary">Uso Temporário</SelectItem>
                </SelectContent>
            </Select>
        </div>
        <div>
            <Label htmlFor="time_of_day">Período do Dia</Label>
            <Select value={formData.time_of_day} onValueChange={v => setFormData({...formData, time_of_day: v})}>
                <SelectTrigger id="time_of_day">
                    <SelectValue placeholder="Selecione o período"/>
                </SelectTrigger>
                <SelectContent>
                    <SelectItem value="morning">Manhã</SelectItem>
                    <SelectItem value="afternoon">Tarde</SelectItem>
                    <SelectItem value="night">Noite</SelectItem>
                    <SelectItem value="dawn">Madrugada</SelectItem>
                    <SelectItem value="any">Qualquer Período</SelectItem>
                </SelectContent>
            </Select>
        </div>
    </div>
    <div className="grid grid-cols-2 gap-4">
        <div>
            <Label htmlFor="start_date">Data de Início *</Label>
            <Input type="date" id="start_date" value={formData.start_date} onChange={e => setFormData({...formData, start_date:
e.target.value})} required />
        </div>
        <div>
            <Label htmlFor="end_date">Data de Fim</Label>
            <Input type="date" id="end_date" value={formData.end_date} onChange={e => setFormData({...formData, end_date:
e.target.value})} />
        </div>
    </div>
    <div>
        <Label htmlFor="prescribing_doctor">Médico Prescritor</Label>
        <Input id="prescribing_doctor" value={formData.prescribing_doctor} onChange={e => setFormData({...formData,
prescribing_doctor: e.target.value})} />
    </div>
    <div>
        <Label htmlFor="indication">Indicação</Label>
        <Input id="indication" value={formData.indication} onChange={e => setFormData({...formData, indication: e.target.value})}
/>
    </div>
    <div>
        <Label htmlFor="side_effects">Efeitos Colaterais</Label>
        <Textarea id="side_effects" value={formData.side_effects} onChange={e => setFormData({...formData, side_effects:
e.target.value})} />
    </div>
    <DialogFooter>
        <Button type="button" variant="outline" onClick={onClose}>Cancelar</Button>
        <Button type="submit" disabled={!isSaving}>{isSaving ? "Salvando..." : "Salvar"}</Button>
    </DialogFooter>
</form>

```

```

    </DialogContent>
  );
};

export default function MedicationsPage() {
  const urlParams = new URLSearchParams(window.location.search);
  const patientIdFromUrl = urlParams.get('patient');

  const [allPatients, setAllPatients] = useState([]);
  const [selectedPatient, setSelectedPatient] = useState(null);
  const [medications, setMedications] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [showForm, setShowForm] = useState(false);
  const [editingMedication, setEditingMedication] = useState(null);

  useEffect(() => {
    loadInitialData();
  }, []);

  useEffect(() => {
    if(selectedPatient) {
      loadMedicationsForPatient(selectedPatient.id);
    } else {
      setMedications([]);
    }
  }, [selectedPatient]);

  const loadInitialData = async () => {
    setIsLoading(true);
    try {
      // Assuming Patient.list() fetches all patients accessible to the current user
      const patients = await Patient.list();
      setAllPatients(patients);

      if (patientIdFromUrl) {
        const patient = patients.find(p => p.id === patientIdFromUrl);
        if(patient) setSelectedPatient(patient);
        else if (patients.length > 0) setSelectedPatient(patients[0]);
        else setSelectedPatient(null);
      } else if (patients.length > 0) {
        setSelectedPatient(patients[0]);
      } else {
        setSelectedPatient(null);
      }
    } catch(e) {
      console.error("Error loading initial data:", e);
      setAllPatients([]);
      setSelectedPatient(null);
    } finally {
      setIsLoading(false);
    }
  }

  const loadMedicationsForPatient = async (patientId) => {
    setIsLoading(true);
    try {
      const medicationData = await MedicationEntity.filter({ patient_id: patientId }, '-start_date');
      setMedications(medicationData);
    } catch (error) {
      console.error("Erro ao carregar medicações:", error);
      setMedications([]);
    } finally {
      setIsLoading(false);
    }
  };

  const handleMedicationCreated = () => {
    if(selectedPatient) loadMedicationsForPatient(selectedPatient.id);
    setEditingMedication(null); // Clear editing state after creation/update
    setShowForm(false); // Close form
  }
}

```

```

    }

    const handlePatientChange = (patientId) => {
      const patient = allPatients.find(p => p.id === patientId);
      setSelectedPatient(patient);
    }

    const handleEdit = (med) => {
      setEditingMedication(med);
      setShowForm(true);
    }

    const handleDelete = async (medId) => {
      if(window.confirm("Tem certeza que deseja apagar esta medicação? Esta ação é irreversível.")) {
        try {
          await MedicationEntity.delete(medId);
          if(selectedPatient) {
            loadMedicationsForPatient(selectedPatient.id);
          }
        } catch (error) {
          console.error("Erro ao apagar medicação:", error);
          alert("Falha ao apagar medicação. Tente novamente.");
        }
      }
    }

    const activeMeds = medications.filter(m => m.status === 'active');
    const pastMeds = medications.filter(m => m.status !== 'active');

    if (isLoading && medications.length === 0 && !selectedPatient && allPatients.length === 0) return <p
className="p-6">Carregando...</p>;

    return (
      <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">
        <Dialog open={showForm} onOpenChange={(isOpen) => {
          if(!isOpen) {
            setEditingMedication(null); // Clear editing state when dialog closes
          }
          setShowForm(isOpen);
        }}>
          <MedicationForm
            isOpen={showForm}
            onClose={() => { setShowForm(false); setEditingMedication(null); }}
            patientId={selectedPatient?.id}
            onMedicationCreated={handleMedicationCreated}
            medicationToEdit={editingMedication}
          />
        </Dialog>

        <div className="max-w-7xl mx-auto space-y-6">
          <div className="flex flex-col md:flex-row justify-between items-center gap-4">
            <div>
              <h1 className="text-3xl font-bold text-gray-900">Medicações de {selectedPatient?.full_name || 'Paciente'}</h1>
              <p className="text-gray-600">Controle seus medicamentos atuais e passados.</p>
            </div>
            <div className="flex items-center gap-4 w-full md:w-auto">
              <Select onValueChange={handlePatientChange} value={selectedPatient?.id || ""}>
                <SelectTrigger className="w-full md:w-[280px]">
                  <SelectValue placeholder="Selecione um paciente..." />
                </SelectTrigger>
                <SelectContent>
                  {allPatients.length > 0 ? (
                    allPatients.map(p => <SelectItem key={p.id} value={p.id}>{p.full_name}</SelectItem>)
                  ) : (
                    <SelectItem disabled value="">Nenhum paciente encontrado</SelectItem>
                  )}
                </SelectContent>
              </Select>
              <Button onClick={() => setShowForm(true)} disabled={!selectedPatient} className="bg-gradient-to-r from-blue-500
to-blue-600 flex-shrink-0">

```

```

        <Plus className="w-4 h-4 mr-2" />
        Adicionar Medicação
      </Button>
    </div>
  </div>

  {!selectedPatient && (
    <Alert className="mt-6"><AlertDescription>
      {allPatients.length > 0 ? "Selecione um paciente para visualizar as medicações." : "Nenhum paciente encontrado. Crie
um perfil de paciente para começar."}
    </AlertDescription></Alert>
  )}

  {selectedPatient && (
    <>
      <Card className="shadow-lg border-0">
        <CardHeader>
          <CardTitle className="flex items-center gap-3 text-green-600">
            <Pill className="w-6 h-6" />
            Medicações em Uso
          </CardTitle>
        </CardHeader>
        <CardContent className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
          {activeMeds.length > 0 ? activeMeds.map((med => (
            <Card key={med.id} className="bg-green-50/50 flex flex-col justify-between">
              <CardHeader>
                <CardTitle className="text-base">{med.commercial_name}</CardTitle>
                <p className="text-sm text-gray-500">{med.generic_name}</p>
              </CardHeader>
              <CardContent className="space-y-2 text-sm">
                <p><span className="font-semibold">Dosagem:</span> {med.dosage}</p>
                <p><span className="font-semibold">Frequência:</span> {med.frequency}</p>
                <p><span className="font-semibold">Início:</span> {format(new Date(med.start_date), "dd/MM/yyyy", {
locale: ptBR })}</p>
                {med.usage_type && <Badge variant="secondary">{med.usage_type === 'continuous' ? 'Uso Contínuo' :
'Uso Temporário'}</Badge>}
              </CardContent>
            <div className="p-4 flex gap-2">
              <Button size="sm" variant="outline" onClick={() => handleEdit(med)}><Edit className="w-3 h-3 mr-1"/>
              <Button size="sm" variant="destructive" onClick={() => handleDelete(med.id)}><Trash2 className="w-3
h-3 mr-1"/> Apagar</Button>
            </div>
          </Card>
        </CardContent>
      </Card>
    </>
  ) : <p className="text-gray-500 col-span-full">Nenhuma medicação em uso.</p>}
  </CardContent>
</Card>

<Card className="shadow-lg border-0">
  <CardHeader>
    <CardTitle className="flex items-center gap-3 text-gray-600">
      <Archive className="w-6 h-6" />
      Histórico de Medicações
    </CardTitle>
  </CardHeader>
  <CardContent className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
    {pastMeds.length > 0 ? pastMeds.map((med => (
      <Card key={med.id}>
        <CardHeader>
          <CardTitle className="text-base">{med.commercial_name}</CardTitle>
          <p className="text-sm text-gray-500">{med.generic_name}</p>
        </CardHeader>
        <CardContent className="space-y-2 text-sm">
          <p><span className="font-semibold">Dosagem:</span> {med.dosage}</p>
          <p><span className="font-semibold">Frequência:</span> {med.frequency}</p>
          <p><span className="font-semibold">Período:</span> {format(new Date(med.start_date), "dd/MM/yy", {
locale: ptBR })} - {med.end_date ? format(new Date(med.end_date), "dd/MM/yy", { locale: ptBR }) : 'Em uso'}</p>
          <Badge variant="secondary">{med.status}</Badge>
          {med.usage_type && <Badge variant="secondary" className="ml-1">{med.usage_type === 'continuous' ?
'Uso Contínuo' : 'Uso Temporário'}</Badge>}
        </CardContent>
      </Card>
    ) : <p className="text-gray-500 col-span-full">Nenhuma medicação em uso.</p>)}
  </CardContent>
</Card>

```

```

        </CardContent>
        <div className="p-4 flex gap-2">
            <Button size="sm" variant="outline" onClick={() => handleEdit(med)}><Edit className="w-3 h-3 mr-1"/>
Editar</Button>
            <Button size="sm" variant="destructive" onClick={() => handleDelete(med.id)}><Trash2 className="w-3
h-3 mr-1"/> Apagar</Button>
        </div>
    </Card>
    )) : <p className="text-gray-500 col-span-full">Nenhum histórico encontrado.</p>
  </CardContent>
</Card>
</>
  )}
</div>
</div>
);
}

```

## pages/Appointments.js

```

import React, { useState, useEffect } from "react";
import { Patient } from "@entities/Patient";
import { Appointment as AppointmentEntity } from "@entities/Appointment";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { Button } from "@components/ui/button";
import { Alert, AlertDescription } from "@components/ui/alert";
import { Plus, Calendar, CheckCircle, Edit, Trash2 } from "lucide-react";
import { format } from "date-fns";
import { ptBR } from "date-fns/locale";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@components/ui/select";
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogFooter } from "@components/ui/dialog";
import { Input } from "@components/ui/input";
import { Label } from "@components/ui/label";

const AppointmentForm = ({ isOpen, onClose, patientId, onAppointmentCreated, appointmentToEdit }) => {
  const [formData, setFormData] = useState({ date: "", doctor: "", specialty: "", location: "", location_url: "" });
  const [isSaving, setIsSaving] = useState(false);

  useEffect(() => {
    if (appointmentToEdit) {
      setFormData({
        ...appointmentToEdit,
        date: appointmentToEdit.date ? format(new Date(appointmentToEdit.date), "yyyy-MM-dd'T'HH:mm") : ""
      });
    } else {
      setFormData({ date: "", doctor: "", specialty: "", location: "", location_url: "" });
    }
  }, [appointmentToEdit]);

  // The Dialog component now controls visibility; no need for `if (!isOpen) return null;` here.

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSaving(true);
    try {
      const dataToSave = { ...formData, patient_id: patientId, status: "scheduled" };
      if (appointmentToEdit) {
        await AppointmentEntity.update(appointmentToEdit.id, dataToSave);
      } else {
        await AppointmentEntity.create(dataToSave);
      }
      onAppointmentCreated();
      onClose();
    } catch (error) {
      console.error("Failed to save appointment:", error);
      // In a real app, you'd show a user-friendly error message here.
    }
  }

```



```

    } finally {
      setIsSaving(false);
    }
  };

  return (
    <DialogContent>
      <DialogHeader>
        <DialogTitle>{appointmentToEdit ? 'Editar' : 'Agendar'} Consulta</DialogTitle>
      </DialogHeader>
      <form onSubmit={handleSubmit} className="space-y-4">
        <div>
          <Label htmlFor="date">Data e Hora</Label>
          <Input id="date" type="datetime-local" value={formData.date} onChange={e => setFormData({...formData, date:
e.target.value})} required />
        </div>
        <div>
          <Label htmlFor="doctor">Médico</Label>
          <Input id="doctor" type="text" value={formData.doctor} onChange={e => setFormData({...formData, doctor: e.target.value})}
required />
        </div>
        <div>
          <Label htmlFor="specialty">Especialidade</Label>
          <Input id="specialty" type="text" value={formData.specialty} onChange={e => setFormData({...formData, specialty:
e.target.value})} required />
        </div>
        <div>
          <Label htmlFor="location">Local (Endereço)</Label>
          <Input id="location" type="text" value={formData.location} onChange={e => setFormData({...formData, location:
e.target.value})} />
        </div>
        <div>
          <Label htmlFor="location_url">Link do Mapa (Google Maps)</Label>
          <Input id="location_url" type="url" value={formData.location_url} onChange={e => setFormData({...formData, location_url:
e.target.value})} placeholder="Ex: https://goo.gl/maps/..." />
        </div>
        <DialogFooter>
          <Button type="button" variant="outline" onClick={onClose} disabled={isSaving}>Cancelar</Button>
          <Button type="submit" disabled={isSaving}>{isSaving ? "Salvando..." : "Salvar"}</Button>
        </DialogFooter>
      </form>
    </DialogContent>
  );
};

export default function AppointmentsPage() {
  const urlParams = new URLSearchParams(window.location.search);
  const patientIdFromUrl = urlParams.get('patient');

  const [allPatients, setAllPatients] = useState([]);
  const [selectedPatient, setSelectedPatient] = useState(null);
  const [appointments, setAppointments] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [showForm, setShowForm] = useState(false);
  const [editingAppointment, setEditingAppointment] = useState(null);

  useEffect(() => {
    loadInitialData();
  }, []);

  useEffect(() => {
    if(selectedPatient) {
      loadAppointmentsForPatient(selectedPatient.id);
    } else {
      setAppointments([]);
    }
  }, [selectedPatient]);

  const loadInitialData = async () => {
    setIsLoading(true);

```

```

    try {
      const patients = await Patient.list();
      setAllPatients(patients);
      if (patientIdFromUrl) {
        const patient = patients.find(p => p.id === patientIdFromUrl);
        if(patient) setSelectedPatient(patient);
      } else if (patients.length > 0) {
        setSelectedPatient(patients[0]);
      }
    } catch(e) {
      console.error("Erro ao carregar dados iniciais:", e);
    } finally {
      setIsLoading(false);
    }
  }
}

const loadAppointmentsForPatient = async (patientId) => {
  setIsLoading(true);
  try {
    // Sort by date ascending to easily filter upcoming and past
    const appointmentData = await AppointmentEntity.filter({ patient_id: patientId }, 'date');
    setAppointments(appointmentData);
  } catch (error) {
    console.error("Erro ao carregar consultas:", error);
    setAppointments([]);
  } finally {
    setIsLoading(false);
  }
};

const handleAppointmentCreated = () => {
  if(selectedPatient) loadAppointmentsForPatient(selectedPatient.id);
  setShowForm(false); // Close form after successful operation
  setEditingAppointment(null); // Clear editing state
}

const handlePatientChange = (patientId) => {
  const patient = allPatients.find(p => p.id === patientId);
  setSelectedPatient(patient);
  setEditingAppointment(null); // Clear editing state when patient changes
  setShowForm(false); // Close form if open
}

const handleEdit = (appt) => {
  setEditingAppointment(appt);
  setShowForm(true);
}

const handleDelete = async (apptId) => {
  if(window.confirm("Tem certeza que deseja apagar esta consulta?")) {
    try {
      await AppointmentEntity.delete(apptId);
      if(selectedPatient) loadAppointmentsForPatient(selectedPatient.id);
    } catch (error) {
      console.error("Erro ao apagar consulta:", error);
      // In a real app, you'd show a user-friendly error message here.
    }
  }
}

// Filter and sort appointments
const now = new Date();
const upcoming = appointments.filter(a => new Date(a.date) >= now && a.status === 'scheduled')
  .sort((a, b) => new Date(a.date).getTime() - new Date(b.date).getTime()); // Ascending for upcoming
const past = appointments.filter(a => new Date(a.date) < now || a.status !== 'scheduled')
  .sort((a, b) => new Date(b.date).getTime() - new Date(a.date).getTime()); // Descending for past

if (isLoading && appointments.length === 0 && !selectedPatient) return <p className="p-6 text-center text-gray-500">Carregando...</p>;

```

```

return (
  <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">
    <Dialog open={showForm} onOpenChange={isOpen} => {
      if(!isOpen) { // When dialog closes
        setEditingAppointment(null); // Clear editing state
      }
      setShowForm(isOpen); // Update showForm state
    }>
    <AppointmentForm
      isOpen={showForm}
      onClose={() => { setShowForm(false); setEditingAppointment(null); }}
      patientId={selectedPatient?.id}
      onAppointmentCreated={handleAppointmentCreated}
      appointmentToEdit={editingAppointment}
    />
  </Dialog>
  <div className="max-w-7xl mx-auto space-y-6">
    <div className="flex flex-col md:flex-row justify-between items-center gap-4">
      <div>
        <h1 className="text-3xl font-bold text-gray-900">Consultas de {selectedPatient?.full_name || 'Paciente'}</h1>
        <p className="text-gray-600">Acompanhe suas consultas agendadas e passadas.</p>
      </div>
      <div className="flex items-center gap-4">
        <Select onValueChange={handlePatientChange} value={selectedPatient?.id || ""}>
          <SelectTrigger className="w-[280px]">
            <SelectValue placeholder="Selecione um paciente..." />
          </SelectTrigger>
          <SelectContent>
            {allPatients.length > 0 ? (
              allPatients.map(p => <SelectItem key={p.id} value={p.id}>{p.full_name}</SelectItem>)
            ) : (
              <SelectItem disabled value="">Nenhum paciente encontrado</SelectItem>
            )}
          </SelectContent>
        </Select>
        <Button onClick={() => { setShowForm(true); setEditingAppointment(null); }} disabled={!selectedPatient}>
          Agendar Consulta
        </Button>
      </div>
    </div>

    {!selectedPatient && (
      <Alert className="m-6">
        <AlertDescription>Selecione um paciente para visualizar as consultas ou agendar uma nova.</AlertDescription>
      </Alert>
    )}

    {selectedPatient && (
      <>
        <Card className="shadow-lg border-0">
          <CardHeader><CardTitle className="flex items-center gap-3 text-indigo-600"><Calendar className="w-6 h-6" />Próximas Consultas</CardTitle></CardHeader>
          <CardContent className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
            {upcoming.length > 0 ? upcoming.map(appt => (
              <Card key={appt.id} className="bg-indigo-50/50">
                <CardContent className="p-4 space-y-2">
                  <div>
                    <p className="font-bold text-indigo-800">{format(new Date(appt.date), "dd 'de' MMMM 'de' yyyy 'às' HH:mm", { locale: ptBR })}</p>
                    <p className="font-semibold">{appt.specialty} com Dr(a). {appt.doctor}</p>
                    <p className="text-sm text-gray-600">
                      {appt.location}
                      {appt.location_url && (
                        <a href__={appt.location_url} target="_blank" rel="noopener noreferrer" className="text-blue-500 hover:underline ml-1">
                          (Ver no Mapa)
                        </a>
                      )}
                    </p>
                  </div>
                </CardContent>
              </Card>
            )}
          </CardContent>
        </Card>
      </>
    )}
  </div>

```

```

        </p>
      </div>
      <div className="flex gap-2">
        <Button size="sm" variant="outline" onClick={() => handleEdit(appt)}><Edit className="w-3 h-3 mr-1"/>
      </div>
      <Button size="sm" variant="destructive" onClick={() => handleDelete(appt.id)}><Trash2 className="w-3 h-3 mr-1"/>
    </div>
  </CardContent>
</Card>
)) : <p className="text-gray-500 col-span-full">Nenhuma consulta agendada.</p>
</CardContent>
</Card>

<Card className="shadow-lg border-0">
  <CardHeader><CardTitle className="flex items-center gap-3 text-gray-600"><CheckCircle className="w-6 h-6" />Histórico de Consultas</CardTitle></CardHeader>
  <CardContent className="space-y-2">
    {past.length > 0 ? past.map(appt => (
      <div key={appt.id} className="p-3 border rounded-lg">
        <p className="font-semibold">
          {format(new Date(appt.date), "dd/MM/yyyy 'às' HH:mm", { locale: ptBR })} - {appt.specialty} com Dr(a).
        </p>
        <p className="text-sm text-gray-600">
          {appt.location}
          {appt.location_url && (
            <a href={appt.location_url} target="_blank" rel="noopener noreferrer" className="text-blue-500
              hover:underline ml-1">
              (Ver no Mapa)
            </a>
          )}
        </p>
        {appt.summary && <p className="text-sm text-gray-600 mt-1">Resumo: {appt.summary}</p>}
      </div>
    )} : <p className="text-gray-500">Nenhum histórico de consultas.</p>
  </CardContent>
</Card>
</>
  )}
</div>
</div>
);
}

```

## pages/MedicalRequests.js

```

import React, { useState, useEffect } from "react";
import { Patient } from "@entities/Patient";
import { MedicalRequest as MedicalRequestEntity } from "@entities/MedicalRequest";
import { UploadFile } from "@integrations/Core";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { Label } from "@components/ui/label";
import { Textarea } from "@components/ui/textarea";
import { Alert, AlertDescription } from "@components/ui/alert";
import { Plus, ClipboardList, CheckCircle, Upload, FileText, Imagemcon, X, Save } from "lucide-react";
import { format } from "date-fns";
import { ptBR } from "date-fns/locale";
import { Badge } from "@components/ui/badge";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@components/ui/select";
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogTrigger, DialogFooter } from "@components/ui/dialog";

```

```

const RequestForm = ({ isOpen, onClose, patientId, onRequestCreated, requestToEdit }) => {
  const [formData, setFormData] = useState({

```

```

    type: "exam_request",
    title: "",
    requesting_doctor: "",
    request_date: "",
    status: "pending",
    files: []
  });
  const [filesToUpload, setFilesToUpload] = useState([]);
  const [isSaving, setIsSaving] = useState(false);

  useEffect(() => {
    if(requestToEdit) {
      setFormData({
        ...requestToEdit,
        request_date: requestToEdit.request_date ? format(new Date(requestToEdit.request_date), 'yyyy-MM-dd') : ""
      });
    } else {
      setFormData({ type: "exam_request", title: "", requesting_doctor: "", request_date: "", status: "pending", files: [] });
    }
  }, [requestToEdit]);

  if (!isOpen) return null;

  const handleFileChange = (e) => {
    setFilesToUpload([...e.target.files]);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSaving(true);
    try {
      let uploadedFiles = formData.files || [];
      if (filesToUpload.length > 0) {
        const uploadPromises = filesToUpload.map(file => UploadFile({ file }));
        const results = await Promise.all(uploadPromises);
        uploadedFiles = [...uploadedFiles, ...results.map(r => r.file_url)];
      }

      const dataToSave = { ...formData, patient_id: patientId, files: uploadedFiles };

      if (requestToEdit) {
        await MedicalRequestEntity.update(requestToEdit.id, dataToSave);
      } else {
        await MedicalRequestEntity.create(dataToSave);
      }
      onRequestCreated();
      onClose();
    } catch (error) {
      console.error("Failed to save request:", error);
    } finally {
      setIsSaving(false);
    }
  };

  return (
    <DialogContent>
      <DialogHeader><DialogTitle>{requestToEdit ? 'Editar' : 'Novo'} Pedido Médico</DialogTitle></DialogHeader>
      <form onSubmit={handleSubmit} className="space-y-4">
        <div><Label>Tipo</Label><Select value={formData.type} onChange={value => setFormData({...formData, type: value})}><SelectTrigger><SelectValue/></SelectTrigger><SelectContent><SelectItem value="exam_request">Pedido de Exame</SelectItem><SelectItem value="referral">Encaminhamento</SelectItem><SelectItem value="prescription">Prescrição</SelectItem></SelectContent></Select></div>
        <div><Label>Título</Label><Input type="text" value={formData.title} onChange={e => setFormData({...formData, title: e.target.value})} required /></div>
        <div><Label>Médico Solicitante</Label><Input type="text" value={formData.requesting_doctor} onChange={e => setFormData({...formData, requesting_doctor: e.target.value})} required /></div>
        <div><Label>Data</Label><Input type="date" value={formData.request_date} onChange={e => setFormData({...formData, request_date: e.target.value})} required /></div>
        <div><Label>Anexar Pedido (Foto/PDF)</Label><Input type="file" multiple onChange={handleFileChange} /></div>
      </form>
    </DialogContent>
  );

```

```

        <DialogFooter><Button type="button" variant="outline" onClick={onClose}>Cancelar</Button><Button type="submit"
disabled={isSaving}>{isSaving ? "Salvando..." : "Salvar"}</Button></DialogFooter>
    </form>
</DialogContent>
);
};

```

```

export default function MedicalRequestsPage() {
    const urlParams = new URLSearchParams(window.location.search);
    const patientIdFromUrl = urlParams.get('patient');

```

```

    const [allPatients, setAllPatients] = useState([]);
    const [selectedPatient, setSelectedPatient] = useState(null);
    const [requests, setRequests] = useState([]);
    const [isLoading, setIsLoading] = useState(true);
    const [showForm, setShowForm] = useState(false);
    const [editingRequest, setEditingRequest] = useState(null);

```

```

    useEffect(() => {
        loadInitialData();
    }, []);

```

```

    useEffect(() => {
        if(selectedPatient) {
            loadRequestsForPatient(selectedPatient.id);
        } else {
            setRequests([]);
        }
    }, [selectedPatient]);

```

```

    const loadInitialData = async () => {
        setIsLoading(true);
        try {
            const patients = await Patient.list('-created_date');
            setAllPatients(patients);
            if (patientIdFromUrl) {
                const patient = patients.find(p => p.id === patientIdFromUrl);
                if(patient) setSelectedPatient(patient);
                else if (patients.length > 0) setSelectedPatient(patients[0]);
            } else if (patients.length > 0) {
                setSelectedPatient(patients[0]);
            }
        } catch(e) {
            console.error("Erro ao carregar dados iniciais:", e);
        } finally {
            setIsLoading(false);
        }
    }

```

```

    const loadRequestsForPatient = async (patientId) => {
        setIsLoading(true);
        try {
            const requestData = await MedicalRequestEntity.filter({ patient_id: patientId }, '-request_date');
            setRequests(requestData);
        } catch (error) {
            console.error("Erro ao carregar pedidos:", error);
            setRequests([]);
        } finally {
            setIsLoading(false);
        }
    };

```

```

    const handleRequestCreated = () => {
        if(selectedPatient) loadRequestsForPatient(selectedPatient.id);
    }

```

```

    const handlePatientChange = (patientId) => {
        const patient = allPatients.find(p => p.id === patientId);
        setSelectedPatient(patient);
    }

```

```

const handleEdit = (request) => {
  setEditingRequest(request);
  setShowForm(true);
}

const handleDelete = async (requestId) => {
  if(window.confirm("Tem certeza que deseja apagar este pedido?")) {
    await MedicalRequestEntity.delete(requestId);
    loadRequestsForPatient(selectedPatient.id);
  }
}

if (isLoading && requests.length === 0) return <p className="p-6">Carregando...</p>;

return (
  <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">
    <Dialog open={showForm} onOpenChange={isOpen => { if(!isOpen) setEditingRequest(null); setShowForm(isOpen); }}>
      <RequestForm
        isOpen={showForm}
        onClose={() => { setShowForm(false); setEditingRequest(null); }}
        patientId={selectedPatient?.id}
        onRequestCreated={handleRequestCreated}
        requestToEdit={editingRequest}
      />
    </Dialog>
    <div className="max-w-7xl mx-auto space-y-6">
      <div className="flex flex-col md:flex-row justify-between items-center gap-4">
        <div>
          <h1 className="text-3xl font-bold text-gray-900">Pedidos Médicos de {selectedPatient?.full_name || 'Paciente'}</h1>
          <p className="text-gray-600">Acompanhe seus pedidos, receitas e encaminhamentos.</p>
        </div>
        <div className="flex items-center gap-4">
          <Select onChange={handlePatientChange} value={selectedPatient?.id || ""}>
            <SelectTrigger className="w-[280px]">
              <SelectValue placeholder="Selecione um paciente..." />
            </SelectTrigger>
            <SelectContent>
              {allPatients.length > 0 ? (
                allPatients.map(p => <SelectItem key={p.id} value={p.id}>{p.full_name}</SelectItem>)
              ) : (
                <SelectItem disabled value="no-patients">Nenhum paciente disponível</SelectItem>
              )}
            </SelectContent>
          </Select>
          <Button onClick={() => setShowForm(true)} disabled={!selectedPatient} className="bg-gradient-to-r from-blue-500 to-blue-600">
            <Plus className="w-4 h-4 mr-2" />
            Adicionar Pedido
          </Button>
        </div>
      </div>

      {!selectedPatient && <Alert className="mt-6"><AlertDescription>Selecione um paciente para visualizar os pedidos.</AlertDescription></Alert>}

      {selectedPatient && (
        <Card className="shadow-lg border-0">
          <CardHeader><CardTitle className="flex items-center gap-3 text-teal-600"><ClipboardList className="w-6 h-6" />Todos os Pedidos</CardTitle></CardHeader>
          <CardContent className="space-y-4">
            {requests.length > 0 ? (
              requests.map(req => (
                <div key={req.id} className="p-4 border rounded-lg flex justify-between items-center">
                  <div>
                    <p className="font-semibold">{req.title}</p>
                    <p className="text-sm text-gray-500">{format(new Date(req.request_date), "dd/MM/yyyy", { locale: ptBR })} - Dr(a). {req.requesting_doctor}</p>
                    {req.files && req.files.length > 0 && <a href__={req.files[0]} target="_blank" className="text-sm text-blue-600">Ver anexo</a>}
                  </div>
                </div>
              )
            )}
          </CardContent>
        </Card>
      )}
    </div>
  </div>

```

```

        </div>
        <div className="flex items-center gap-2">
          <Badge variant={req.status === 'pending' ? 'destructive' : 'default'}>{req.status}</Badge>
          <Button variant="outline" size="sm" onClick={() => handleEdit(req)}>Editar</Button>
          <Button variant="destructive" size="sm" onClick={() => handleDelete(req.id)}>Apagar</Button>
        </div>
      </div>
    ))
  ) : (
    <p className="text-gray-500">Nenhum pedido médico encontrado para este paciente.</p>
  )
}
</CardContent>
</Card>
  ))
</div>
</div>
);
}

```

## pages/Search.js

```

import React, { useState, useEffect } from "react";
import { User } from "@entities/User";
import { Patient } from "@entities/Patient";
import { Exam } from "@entities/Exam";
import { Medication } from "@entities/Medication";
import { Appointment } from "@entities/Appointment";
import { MedicalHistory } from "@entities/MedicalHistory";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { Input } from "@components/ui/input";
import { Search as SearchIcon, FileText, Pill, Calendar, HeartPulse } from "lucide-react";
import { format } from "date-fns";
import { ptBR } from "date-fns/locale";

export default function SearchPage() {
  const [patient, setPatient] = useState(null);
  const [searchTerm, setSearchTerm] = useState("");
  const [results, setResults] = useState(null);
  const [isLoading, setIsLoading] = useState(false);

  useEffect(() => {
    const loadPatient = async () => {
      const user = await User.me();
      const patients = await Patient.filter({ created_by: user.email });
      if (patients.length > 0) {
        setPatient(patients[0]);
      }
    };
    loadPatient();
  }, []);

  const handleSearch = async (e) => {
    e.preventDefault();
    if (!searchTerm || !patient) return;

    setIsLoading(true);
    setResults(null);

    const [exams, medications, appointments, history] = await Promise.all([
      Exam.filter({ patient_id: patient.id }),
      Medication.filter({ patient_id: patient.id }),
      Appointment.filter({ patient_id: patient.id }),
      MedicalHistory.filter({ patient_id: patient.id })
    ]);

    const lowerSearchTerm = searchTerm.toLowerCase();
  }

```



```

const filter = (items, fields) => items.filter(item =>
  fields.some(field =>
    item[field] && item[field].toString().toLowerCase().includes(lowerSearchTerm)
  )
);

setResults({
  exams: filter(exams, ['exam_type', 'requesting_doctor', 'observations']),
  medications: filter(medications, ['commercial_name', 'generic_name', 'indication']),
  appointments: filter(appointments, ['doctor', 'specialty', 'summary']),
  history: filter(history, ['title', 'description', 'doctor'])
});

setIsLoading(false);
};

const ResultCard = ({ title, icon: Icon, color, data, renderItem }) => (
  <Card>
    <CardHeader><CardTitle className={`flex items-center gap-2 ${color}`}><Icon className="w-5 h-5"
/><title></CardTitle></CardHeader>
    <CardContent className="space-y-2">
      {data.length > 0 ? data.map(renderItem) : <p className="text-sm text-gray-500">Nenhum resultado encontrado.</p>}
    </CardContent>
  </Card>
);

return (
  <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-6">
    <div className="max-w-4xl mx-auto space-y-6">
      <div>
        <h1 className="text-3xl font-bold text-gray-900">Busca Rápida</h1>
        <p className="text-gray-600">Encontre qualquer informação no seu prontuário.</p>
      </div>

      <form onSubmit={handleSearch} className="relative">
        <SearchIcon className="absolute left-3 top-1/2 -translate-y-1/2 text-gray-400" />
        <Input
          value={searchTerm}
          onChange={e => setSearchTerm(e.target.value)}
          placeholder="Pesquisar exames, medicações, médicos..."
          className="pl-10 h-12 text-lg"
        />
      </form>

      {isLoading && <p>Buscando...</p>}

      {results && (
        <div className="space-y-4">
          <ResultCard
            title="Exames" icon={FileText} color="text-orange-500" data={results.exams}
            renderItem={item => <div key={item.id} className="p-2 border rounded"><p>{item.exam_type} - {format(new
Date(item.exam_date), "dd/MM/yy")}</p></div>}
          />
          <ResultCard
            title="Medicações" icon={Pill} color="text-red-500" data={results.medications}
            renderItem={item => <div key={item.id} className="p-2 border rounded"><p>{item.commercial_name}
({item.dosage})</p></div>}
          />
          <ResultCard
            title="Consultas" icon={Calendar} color="text-indigo-500" data={results.appointments}
            renderItem={item => <div key={item.id} className="p-2 border rounded"><p>{item.specialty} com Dr(a) {item.doctor}
- {format(new Date(item.date), "dd/MM/yy")}</p></div>}
          />
          <ResultCard
            title="Histórico Médico" icon={HeartPulse} color="text-red-500" data={results.history}
            renderItem={item => <div key={item.id} className="p-2 border rounded"><p>{item.title} - {format(new
Date(item.date), "dd/MM/yy")}</p></div>}
          />
        </div>
      )}
    </div>
  </div>
);

```

```

    </div>
  </div>
);
}

```

## pages/PatientDetails.js

```

import React, { useState, useEffect } from "react";
import { Patient } from "@entities/Patient";
import { Exam } from "@entities/Exam";
import { Medication } from "@entities/Medication";
import { Appointment } from "@entities/Appointment";
import { MedicalHistory } from "@entities/MedicalHistory";
import { Credential } from "@entities/Credential";
import { Incident } from "@entities/Incident";
import { MedicalRequest } from "@entities/MedicalRequest";
import { UploadFile } from "@integrations/Core";
import { Card, CardContent, CardHeader, CardTitle } from "@components/ui/card";
import { Button } from "@components/ui/button";
import { Badge } from "@components/ui/badge";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@components/ui/tabs";
import {
  ArrowLeft,
  Edit,
  User,
  TestTube,
  Pill,
  Calendar,
  FileText,
  KeyRound,
  FileImage,
  Plus,
  Eye,
  EyeOff,
  Trash2,
  ChevronRight,
  Copy,
  ExternalLink,
  Lock,
  Unlock,
  ClipboardList,
  Phone
} from "lucide-react";
import { Link, useNavigate } from "react-router-dom";
import { createPageUrl } from "@utils";
import { format, parseISO } from "date-fns";
import { ptBR } from "date-fns/locale";
import { Avatar, AvatarImage, AvatarFallback } from "@components/ui/avatar";
import { Dialog, DialogContent, DialogHeader, DialogTitle, DialogTrigger, DialogFooter, DialogClose } from "@components/ui/dialog";
import { Input } from "@components/ui/input";
import { Label } from "@components/ui/label";
import { Textarea } from "@components/ui/textarea";

// Form Component for Credentials
const CredentialForm = ({ patientId, onDataAdded }) => {
  const [formData, setFormData] = useState({ patient_id: patientId, service_name: "", url: "", username: "", password: "", notes: "" });
  const [isSaving, setIsSaving] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSaving(true);
    await Credential.create(formData);
    onDataAdded();
    setIsSaving(false);
    setFormData({ patient_id: patientId, service_name: "", url: "", username: "", password: "", notes: "" });
  };

```

```

return (
  <form onSubmit={handleSubmit} className="space-y-4">
    <div>
      <Label htmlFor="service_name">Nome do Serviço</Label>
      <Input id="service_name" placeholder="Ex: Portal Unimed" value={formData.service_name} onChange={e =>
setFormData({...formData, service_name: e.target.value})} required />
    </div>
    <div>
      <Label htmlFor="url">URL do site</Label>
      <Input id="url" placeholder="https://example.com" value={formData.url} onChange={e => setFormData({...formData, url:
e.target.value})} />
    </div>
    <div>
      <Label htmlFor="username">Usuário</Label>
      <Input id="username" placeholder="Seu Usuário" value={formData.username} onChange={e => setFormData({...formData,
username: e.target.value})} />
    </div>
    <div>
      <Label htmlFor="password">Senha</Label>
      <Input id="password" type="password" placeholder="Sua Senha" value={formData.password} onChange={e =>
setFormData({...formData, password: e.target.value})} />
    </div>
    <div>
      <Label htmlFor="notes">Notas adicionais</Label>
      <Textarea id="notes" placeholder="Informações úteis sobre a credencial" value={formData.notes} onChange={e =>
setFormData({...formData, notes: e.target.value})} />
    </div>
    <DialogFooter>
      <Button type="submit" disabled={isSaving}>{isSaving ? 'Salvando...' : 'Salvar Credencial'}</Button>
    </DialogFooter>
  </form>
);
};

```

```

// Component for Credential Card with Password Visibility
const CredentialCard = ({ credential, onDelete }) => {
  const [showPassword, setShowPassword] = useState(false);

```

```

  const copyToClipboard = (text) => {
    navigator.clipboard.writeText(text);
  };

```

```

return (
  <Card className="p-4 border">
    <div className="space-y-3">
      <div className="flex justify-between items-start">
        <div>
          <h4 className="font-semibold text-gray-900">{credential.service_name}</h4>
          {credential.url && (
            <a
              href__={credential.url.startsWith('http') ? credential.url : `https://${credential.url}`}
              target="_blank"
              rel="noopener noreferrer"
              className="text-sm text-blue-600 hover:text-blue-800 flex items-center gap-1"
            >
              Acessar Site <ExternalLink className="w-3 h-3" />
            </a>
          )}
        </div>
        <Button
          variant="ghost"
          size="icon"
          onClick={() => onDelete(credential.id)}
          className="text-red-500 hover:text-red-700"
        >
          <Trash2 className="w-4 h-4" />
        </Button>
      </div>
      {credential.username && (

```

```

    <div className="flex items-center justify-between bg-gray-50 p-2 rounded">
      <div>
        <span className="text-xs text-gray-500">Usuário:</span>
        <p className="font-mono text-sm">{credential.username}</p>
      </div>
      <Button
        variant="ghost"
        size="sm"
        onClick={() => copyToClipboard(credential.username)}
      >
        <Copy className="w-3 h-3" />
      </Button>
    </div>
  )}

  {credential.password && (
    <div className="flex items-center justify-between bg-gray-50 p-2 rounded">
      <div className="flex-1">
        <span className="text-xs text-gray-500">Senha:</span>
        <p className="font-mono text-sm">
          {showPassword ? credential.password : '.....'}
        </p>
      </div>
      <div className="flex gap-1">
        <Button
          variant="ghost"
          size="sm"
          onClick={() => setShowPassword(!showPassword)}
        >
          {showPassword ? <EyeOff className="w-3 h-3" /> : <Eye className="w-3 h-3" />}
        </Button>
        <Button
          variant="ghost"
          size="sm"
          onClick={() => copyToClipboard(credential.password)}
        >
          <Copy className="w-3 h-3" />
        </Button>
      </div>
    </div>
  )}

  {credential.notes && (
    <div className="bg-blue-50 p-2 rounded">
      <span className="text-xs text-gray-500">Notas:</span>
      <p className="text-sm text-gray-700">{credential.notes}</p>
    </div>
  )}
</div>
</Card>
);
};

// Form Component for Incidents
const IncidentForm = ({ patientId, onDataAdded }) => {
  const [formData, setFormData] = useState({ patient_id: patientId, title: "", description: "", start_date: "", status: 'active' });
  const [isSaving, setIsSaving] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSaving(true);
    await Incident.create(formData);
    onDataAdded();
    setIsSaving(false);
    setFormData({ patient_id: patientId, title: "", description: "", start_date: "", status: 'active' });
  };

  return (
    <form onSubmit={handleSubmit} className="space-y-4">
      <div>

```

```

    <Label htmlFor="title">Título do Incidente</Label>
    <Input id="title" placeholder="Ex: Queda de bicicleta" value={formData.title} onChange={e => setFormData({...formData, title:
e.target.value})} required />
  </div>
  <div>
    <Label htmlFor="start_date">Data de Início</Label>
    <Input id="start_date" type="date" value={formData.start_date} onChange={e => setFormData({...formData, start_date:
e.target.value})} required />
  </div>
  <div>
    <Label htmlFor="description">Descrição Detalhada</Label>
    <Textarea id="description" placeholder="Descreva o incidente..." value={formData.description} onChange={e =>
setFormData({...formData, description: e.target.value})} />
  </div>
  <DialogFooter>
    <Button type="submit" disabled={isSaving}>{isSaving ? 'Salvando...' : 'Salvar Incidente'}</Button>
  </DialogFooter>
</form>
);
};

```

```

// Form Component for Medical Requests / Pendências

```

```

const RequestForm = ({ patientId, onDataAdded, requestToEdit, onClose }) => {
  const [formData, setFormData] = useState({
    patient_id: patientId,
    title: "",
    description: "",
    request_date: new Date().toISOString().split('T')[0],
    status: 'pending',
    files: [],
    requesting_doctor: "",
    type: 'exam_request'
  });
  const [filesToUpload, setFilesToUpload] = useState([]);
  const [isSaving, setIsSaving] = useState(false);

  useEffect(() => {
    if (requestToEdit) {
      setFormData({
        ...requestToEdit,
        request_date: requestToEdit.request_date ? format(parseISO(requestToEdit.request_date), 'yyyy-MM-dd') : "",
      });
    }
  }, [requestToEdit]);

  const handleFileChange = (e) => {
    setFilesToUpload([...e.target.files]);
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSaving(true);
    try {
      let uploadedFiles = formData.files || [];
      if (filesToUpload.length > 0) {
        const uploadPromises = filesToUpload.map(file => UploadFile({ file }));
        const results = await Promise.all(uploadPromises);
        uploadedFiles = [...uploadedFiles, ...results.map(r => r.file_url)];
      }

      const dataToSave = { ...formData, files: uploadedFiles };

      if (requestToEdit) {
        await MedicalRequest.update(requestToEdit.id, dataToSave);
      } else {
        await MedicalRequest.create(dataToSave);
      }
      onDataAdded();
    } catch (error) {
      console.error("Failed to save request:", error);
    }
  };

```

```

    } finally {
      setIsSaving(false);
      onClose();
    }
  };

  return (
    <form onSubmit={handleSubmit} className="space-y-4">
      <div>
        <Label htmlFor="title">Título da Pendência</Label>
        <Input id="title" placeholder="Ex: Marcar Cardiologista" value={formData.title} onChange={e => setFormData({ ...formData, title:
e.target.value })} required />
      </div>
      <div>
        <Label htmlFor="description">Descrição / Observações</Label>
        <Textarea id="description" placeholder="Descreva a pendência..." value={formData.description} onChange={e => setFormData({
...formData, description: e.target.value })} />
      </div>
      <div>
        <Label htmlFor="request_date">Data do Pedido/Prazo</Label>
        <Input id="request_date" type="date" value={formData.request_date} onChange={e => setFormData({ ...formData, request_date:
e.target.value })} required />
      </div>
      <div>
        <Label htmlFor="files">Anexar Pedido/Documento (Opcional)</Label>
        <Input id="files" type="file" multiple onChange={handleFileChange} />
      </div>
      <DialogFooter>
        <Button type="button" variant="outline" onClick={onClose}>Cancelar</Button>
        <Button type="submit" disabled={!isSaving}>{isSaving ? 'Salvando...' : 'Salvar Pendência'}</Button>
      </DialogFooter>
    </form>
  );
};

```

```

export default function PatientDetails() {
  const navigate = useNavigate();
  const urlParams = new URLSearchParams(window.location.search);
  const patientId = urlParams.get('id');

  const [patient, setPatient] = useState(null);
  const [data, setData] = useState({
    exams: [],
    medications: [],
    appointments: [],
    history: [],
    incidents: [],
    credentials: [],
    requests: [],
  });
  const [isLoading, setIsLoading] = useState(true);
  const [isIncidentDialogOpen, setIsIncidentDialogOpen] = useState(false);
  const [isCredentialDialogOpen, setIsCredentialDialogOpen] = useState(false);
  const [isRequestDialogOpen, setIsRequestDialogOpen] = useState(false);
  const [editingRequest, setEditingRequest] = useState(null);

  // State for sensitive data access
  const [sensitivePasswordInput, setSensitivePasswordInput] = useState("");
  const [sensitivePasswordError, setSensitivePasswordError] = useState("");
  const [isSensitiveContentVisible, setIsSensitiveContentVisible] = useState(false);
  const [activeTab, setActiveTab] = useState("overview");

  useEffect(() => {
    if (patientId) {
      loadPatientData();
    }
  }, [patientId]);

  useEffect(() => {

```

```

// Reset unlock status when patient changes, or on initial load if password not active
if (patient) {
  // If no sensitive password is set, or if it's explicitly inactive, then it's "unlocked" by default
  setIsSensitiveContentVisible(!patient.sensitive_data_password_active);
  setActiveTab("overview"); // Always start on a non-sensitive tab
}
}, [patient]);

const loadPatientData = async () => {
  setIsLoading(true);
  try {
    const [
      patientData,
      exams,
      medications,
      appointments,
      history,
      credentials,
      incidents,
      requests
    ] = await Promise.all([
      Patient.filter({id: patientId}),
      Exam.filter({ patient_id: patientId }, '-exam_date', 5),
      Medication.filter({ patient_id: patientId }, '-start_date', 5),
      Appointment.filter({ patient_id: patientId }, '-date', 5),
      MedicalHistory.filter({ patient_id: patientId }, '-date', 5),
      Credential.filter({ patient_id: patientId }, '-created_date'),
      Incident.filter({ patient_id: patientId }, '-start_date', 5),
      MedicalRequest.filter({ patient_id: patientId }, '-request_date')
    ]);

    if (patientData.length > 0) {
      const foundPatient = patientData[0];
      setPatient(foundPatient);

      setData({
        exams,
        medications,
        appointments,
        history,
        credentials,
        incidents,
        requests
      });
    }
  } catch (error) {
    console.error("Erro ao carregar dados do paciente:", error);
  } finally {
    setIsLoading(false);
  }
};

const sensitiveTabs = ["history", "incidents", "credentials", "requests"];

const handleTabChange = (value) => {
  if (sensitiveTabs.includes(value) && !isSensitiveContentVisible) {
    setSensitivePasswordInput("");
    setSensitivePasswordError("");
  }
  setActiveTab(value);
};

const checkPassword = () => {
  if (patient && patient.sensitive_data_password && sensitivePasswordInput === patient.sensitive_data_password) {
    setIsSensitiveContentVisible(true);
    setSensitivePasswordInput("");
    setSensitivePasswordError("");
  } else {
    setSensitivePasswordError("Senha incorreta. Tente novamente.");
  }
};

```

```

    }
  };

  const calculateAge = (birthDate) => {
    if (!birthDate) return "N/A";
    return Math.floor((new Date() - new Date(birthDate)) / (365.25 * 24 * 60 * 60 * 1000));
  };

  const handleOpenRequestDialog = (req = null) => {
    setEditingRequest(req);
    setIsRequestDialogOpen(true);
  }

  const handleDeleteRequest = async (id) => {
    if(window.confirm("Tem certeza que deseja apagar esta pendência?")) {
      await MedicalRequest.delete(id);
      loadPatientData();
    }
  }

  const handleToggleRequestStatus = async (req) => {
    const newStatus = req.status === 'pending' ? 'completed' : 'pending';
    await MedicalRequest.update(req.id, { status: newStatus });
    loadPatientData();
  }

  const handleDataAdded = () => {
    loadPatientData();
    setIsIncidentDialogOpen(false);
    setIsCredentialDialogOpen(false);
    setIsRequestDialogOpen(false);
  };

  if (isLoading) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="flex items-center gap-2">
          <div className="animate-spin rounded-full h-8 w-8 border-b-2 border-blue-600"></div>
          <span className="text-gray-600">Carregando dados do paciente...</span>
        </div>
      </div>
    );
  }

  if (!patient) {
    return (
      <div className="min-h-screen flex items-center justify-center">
        <div className="text-center">
          <h2 className="text-2xl font-bold text-gray-900 mb-2">Paciente não encontrado</h2>
          <Button onClick={() => navigate(createPageUrl("Dashboard"))}>
            <ArrowLeft className="w-4 h-4 mr-2" />
            Voltar ao Dashboard
          </Button>
        </div>
      </div>
    );
  }

  const renderItemCard = (item, title, date, link) => (
    <Link to={link} key={item.id}>
      <Card className="p-3 flex justify-between items-center hover:bg-gray-50 transition-colors">
        <div>
          <p className="font-semibold">{title}</p>
          <p className="text-sm text-gray-500">{date}</p>
        </div>
        <ChevronRight className="w-4 h-4 text-gray-400"/>
      </Card>
    </Link>
  );

```



```

return (
  <div className="min-h-screen bg-gradient-to-br from-slate-50 to-blue-50 p-4 md:p-6">
    <div className="max-w-7xl mx-auto space-y-6">
      {/* Header */}
      <div className="flex flex-col md:flex-row justify-between items-start md:items-center gap-4">
        <div className="flex items-center gap-4">
          <Button
            variant="outline"
            onClick={() => navigate(createPageUrl("Dashboard"))}
          >
            <ArrowLeft className="w-4 h-4 mr-2" />
            Voltar
          </Button>
        </div>
        <h1 className="text-3xl font-bold text-gray-900 mb-2">
          {patient.full_name}
        </h1>
        <p className="text-gray-600">Informações completas do paciente</p>
      </div>
      <div>
        <Link to={createPageUrl(`PatientProfile?id=${patient.id}`)}>
          <Button className="bg-gradient-to-r from-blue-500 to-blue-600 hover:from-blue-600 hover:to-blue-700">
            <Edit className="w-4 h-4 mr-2" />
            Editar Perfil
          </Button>
        </Link>
      </div>
    </div>

    {/* Informações Básicas */}
    <Card className="shadow-lg border-0">
      <CardHeader>
        <CardTitle className="flex items-center gap-2">
          <User className="w-5 h-5 text-blue-600" />
          Informações Pessoais
        </CardTitle>
      </CardHeader>
      <CardContent>
        <div className="grid md:grid-cols-3 gap-6 items-start">
          <div className="text-center">
            <Avatar className="w-24 h-24 mx-auto border-4 border-white shadow-lg">
              <AvatarImage src={patient.photo_url} alt={patient.full_name} />
              <AvatarFallback className="bg-blue-100 text-blue-700 text-3xl">
                {patient.full_name?.charAt(0)?.toUpperCase() || 'P'}
              </AvatarFallback>
            </Avatar>
            <h3 className="font-bold text-xl text-gray-900 mt-4">{patient.full_name}</h3>
            <div className="flex justify-center gap-2 mt-2">
              <Badge variant="outline">{calculateAge(patient.birth_date)} anos</Badge>
              {patient.blood_type && (
                <Badge variant="outline" className="bg-red-50 text-red-700 border-red-200">
                  {patient.blood_type}
                </Badge>
              )}
            </div>
          </div>
        </div>
      </div>

      <div className="space-y-3">
        <div>
          <h4 className="font-semibold text-gray-900 mb-1">Contato de Emergência</h4>
          {patient.emergency_contact_name ? (
            <div>
              <p className="text-gray-700">{patient.emergency_contact_name}</p>
              {patient.emergency_contact_phone && (
                <a href__={tel:patient.emergency_contact_phone} className="text-gray-500 text-sm flex items-center gap-1">
                  <Phone className="w-3 h-3" />
                  {patient.emergency_contact_phone}
                </a>
              )}
            </div>
          ) : (

```

```

        <p className="text-gray-500 text-sm">Não informado</p>
    )}
</div>

<div>
    <h4 className="font-semibold text-gray-900 mb-1">Plano de Saúde</h4>
    {patient.insurance_plan ? (
        <div>
            <p className="text-gray-700">{patient.insurance_plan}</p>
            <p className="text-gray-500 text-sm">
                Cartão: {patient.insurance_number || "N/A"}
            </p>
            {patient.insurance_validity && (
                <p className="text-gray-500 text-sm">
                    Válido até: {format(parseISO(patient.insurance_validity), "dd/MM/yyyy", { locale: ptBR })}
                </p>
            )}
        </div>
    ) : (
        <p className="text-gray-500 text-sm">Não informado</p>
    )}
</div>
</div>

<div className="space-y-3">
    <div>
        <h4 className="font-semibold text-gray-900 mb-1">Médico Responsável</h4>
        {patient.primary_doctor ? (
            <div>
                <p className="text-gray-700">{patient.primary_doctor}</p>
                <p className="text-gray-500 text-sm">{patient.primary_doctor_specialty}</p>
                {patient.primary_doctor_contact && (
                    <p className="text-gray-500 text-sm">{patient.primary_doctor_contact}</p>
                )}
            </div>
        ) : (
            <p className="text-gray-500 text-sm">Não informado</p>
        )}
    </div>

    <div>
        <h4 className="font-semibold text-gray-900 mb-1">Alergias</h4>
        {patient.allergies && patient.allergies.length > 0 ? (
            <div className="flex flex-wrap gap-1">
                {patient.allergies.map((allergy, index) => (
                    <Badge key={index} variant="destructive" className="text-xs">
                        {allergy}
                    </Badge>
                ))}
            </div>
        ) : (
            <p className="text-gray-500 text-sm">Nenhuma alergia registrada</p>
        )}
    </div>
</div>
</CardContent>
</Card>

{ /* Abas de Informações Detalhadas */ }
<Card className="shadow-lg border-0">
    <CardContent className="p-6">
        <Tabs value={activeTab} onChange={handleTabChange}>
            <TabList className="grid w-full grid-cols-3 sm:flex sm:flex-wrap justify-start p-2 h-auto rounded-lg bg-gray-100">
                <TabTrigger value="overview">
                    <TestTube className="w-4 h-4 mr-2"/>Exames
                </TabTrigger>
                <TabTrigger value="medications">
                    <Pill className="w-4 h-4 mr-2"/>Medicações
                </TabTrigger>
            </TabList>
        </Tabs>
    </CardContent>
</Card>

```

```

        <TabsTrigger value="appointments">
        <Calendar className="w-4 h-4 mr-2"/>Consultas
        </TabsTrigger>
        <TabsTrigger value="history">
        {isSensitiveContentVisible ? <Unlock className="w-4 h-4 mr-2 text-green-600"/> : <Lock className="w-4 h-4 mr-2
text-red-600"/>}
        Histórico
        </TabsTrigger>
        <TabsTrigger value="incidents">
        {isSensitiveContentVisible ? <Unlock className="w-4 h-4 mr-2 text-green-600"/> : <Lock className="w-4 h-4 mr-2
text-red-600"/>}
        Incidentes
        </TabsTrigger>
        <TabsTrigger value="requests">
        {isSensitiveContentVisible ? <Unlock className="w-4 h-4 mr-2 text-green-600"/> : <Lock className="w-4 h-4 mr-2
text-red-600"/>}
        Pendências
        </TabsTrigger>
        <TabsTrigger value="credentials">
        {isSensitiveContentVisible ? <Unlock className="w-4 h-4 mr-2 text-green-600"/> : <Lock className="w-4 h-4 mr-2
text-red-600"/>}
        Senhas
        </TabsTrigger>
    </TabsList>

    <TabsContent value="overview" className="mt-4">
        {data.exams.length > 0 ? (
        <div className="space-y-3">
            {data.exams.map(exam => renderItemCard(exam, exam.exam_type, format(parseISO(exam.exam_date),
"dd/MM/yyyy", { locale: ptBR })), createPageUrl('Exams?patient=${patientId}'))}
            </div>
        ) : <p className="text-center text-gray-500 py-4">Nenhum exame recente.</p>
        <Link to={createPageUrl('Exams?patient=${patientId}')}><Button className="mt-4 w-full" variant="outline">Ver Todos os
Exames</Button></Link>
        </TabsContent>

    <TabsContent value="medications" className="mt-4">
        {data.medications.length > 0 ? (
        <div className="space-y-3">
            {data.medications.map(med => renderItemCard(med, `${med.commercial_name} (${med.dosage})`, `Desde:
${format(parseISO(med.start_date), "dd/MM/yyyy", { locale: ptBR })}`, createPageUrl('Medications?patient=${patientId}'))}
            </div>
        ) : <p className="text-center text-gray-500 py-4">Nenhuma medicação recente.</p>
        <Link to={createPageUrl('Medications?patient=${patientId}')}><Button className="mt-4 w-full" variant="outline">Ver
Todas as Medicações</Button></Link>
        </TabsContent>

    <TabsContent value="appointments" className="mt-4">
        {data.appointments.length > 0 ? (
        <div className="space-y-3">
            {data.appointments.map(appt => renderItemCard(appt, `${appt.specialty} com Dr(a). ${appt.doctor}`,
format(parseISO(appt.date), "dd/MM/yyyy 'às' HH:mm", { locale: ptBR })), createPageUrl('Appointments?patient=${patientId}'))}
            </div>
        ) : <p className="text-center text-gray-500 py-4">Nenhuma consulta recente.</p>
        <Link to={createPageUrl('Appointments?patient=${patientId}')}><Button className="mt-4 w-full" variant="outline">Ver
Todas as Consultas</Button></Link>
        </TabsContent>

    {/* Conditional rendering for sensitive tabs */}
    {sensitiveTabs.map(tabId => (
    <TabsContent key={tabId} value={tabId} className="mt-4">
        {isSensitiveContentVisible ? (
        tabId === "history" ? (
        data.history.length > 0 ? (
        <div className="space-y-3">
            {data.history.map(hist => renderItemCard(hist, hist.title, format(parseISO(hist.date), "dd/MM/yyyy", { locale: ptBR })),
createPageUrl('MedicalHistory?patient=${patientId}'))}
            </div>
        ) : <p className="text-center text-gray-500 py-4">Nenhum histórico recente.</p>
        ) : tabId === "incidents" ? (

```

```

<>
<Dialog open={isIncidentDialogOpen} onOpenChange={setIsIncidentDialogOpen}>
  <DialogTrigger asChild>
    <Button className="w-full mb-4"><Plus className="w-4 h-4 mr-2"/>Registrar Novo Incidente</Button>
  </DialogTrigger>
  <DialogContent>
    <DialogHeader><DialogTitle>Novo Incidente</DialogTitle></DialogHeader>
    <IncidentForm patientId={patientId} onDataAdded={handleDataAdded} />
  </DialogContent>
</Dialog>
{data.incidents.length > 0 ? (
  <div className="space-y-3">
    {data.incidents.map(inc => renderItemCard(inc, inc.title, `Início: ${format(parseISO(inc.start_date), "dd/MM/yyyy", {
locale: ptBR }}}`, '#'))}
  </div>
) : <p className="text-center text-gray-500 py-4">Nenhum incidente registrado.</p>
</>
) : tabId === "requests" ? (
  <>
    <Button className="w-full mb-4" onClick={() => handleOpenRequestDialog()}>
    <Plus className="w-4 h-4 mr-2"/>Adicionar Pendência
  </Button>

  <div className="space-y-4">
    <h4 className="font-semibold text-lg text-orange-600">Pendentes</h4>
    {data.requests.filter(r => r.status === 'pending').map(req => (
      <Card key={req.id}>
        <CardContent className="p-3 flex items-center justify-between">
          <div>
            <p className="font-semibold">{req.title}</p>
            <p className="text-sm text-gray-500">
              Pedido em: {format(parseISO(req.request_date), "dd/MM/yyyy", { locale: ptBR }}}
            </p>
            {req.files && req.files.length > 0 && req.files[0] &&
              <a href__={req.files[0]} target="_blank" rel="noopener noreferrer" className="text-blue-600 text-sm
hover:underline">Ver anexo</a>
            }
          </div>
          <div className="flex items-center gap-2">
            <Button size="sm" variant="outline" className="text-green-600 hover:text-green-700 hover:bg-green-50"
onClick={() => handleToggleRequestStatus(req)}>Marcar como Feito</Button>
            <Button size="sm" variant="ghost" onClick={() => handleOpenRequestDialog(req)}><Edit className="w-4
h-4"/></Button>
            <Button size="sm" variant="ghost" className="text-red-600 hover:text-red-700" onClick={() =>
handleDeleteRequest(req.id)}><Trash2 className="w-4 h-4"/></Button>
          </div>
        </CardContent>
      </Card>
    ))}
    {data.requests.filter(r => r.status === 'pending').length === 0 && <p className="text-gray-500 text-center
py-2">Nenhuma pendência.</p>

    <h4 className="font-semibold text-lg text-green-600 pt-4">Concluídas</h4>
    {data.requests.filter(r => r.status === 'completed').map(req => (
      <Card key={req.id} className="bg-gray-50 opacity-70">
        <CardContent className="p-3 flex items-center justify-between">
          <div>
            <p className="font-semibold line-through">{req.title}</p>
            <p className="text-sm text-gray-500">
              Concluído em: {format(parseISO(req.request_date), "dd/MM/yyyy", { locale: ptBR }}}
            </p>
          </div>
          <div className="flex items-center gap-2">
            <Button size="sm" variant="outline" onClick={() => handleToggleRequestStatus(req)}>Marcar como
Pendente</Button>
          </div>
        </CardContent>
      </Card>
    ))}

```

```

        {data.requests.filter(r => r.status === 'completed').length === 0 && <p className="text-gray-500 text-center
py-2">Nenhuma pendência concluída.</p>}
      </div>
    </>
  ) : tabId === "credentials" ? (
    <>
      <Dialog open={isCredentialDialogOpen} onOpenChange={setIsCredentialDialogOpen}>
        <DialogTrigger asChild>
          <Button className="w-full mb-4"><Plus className="w-4 h-4 mr-2"/>Adicionar Nova Senha</Button>
        </DialogTrigger>
        <DialogContent>
          <DialogHeader><DialogTitle>Nova Senha de Acesso</DialogTitle></DialogHeader>
          <CredentialForm patientId={patientId} onDataAdded={handleDataAdded} />
        </DialogContent>
      </Dialog>
      {data.credentials.length > 0 ? (
        <div className="space-y-3">
          {data.credentials.map(cred => (
            <CredentialCard
              key={cred.id}
              credential={cred}
              onDelete={async (id) => {
                await Credential.delete(id);
                loadPatientData();
              }}
            />
          ))}
        </div>
      ) : (
        <p className="text-center text-gray-500 py-4">Nenhuma credencial registrada.</p>
      )}
    </>
  ) : null
) : (
  <div className="text-center p-8">
    <Lock className="w-12 h-12 mx-auto text-gray-400"/>
    <h3 className="mt-4 text-lg font-semibold">Conteúdo Protegido</h3>
    <p className="text-gray-500">Para visualizar estes dados, por favor, insira a senha de acesso sensível do
paciente.</p>
    <div className="mt-4 max-w-sm mx-auto">
      <Input
        type="password"
        placeholder="Senha de acesso"
        value={sensitivePasswordInput}
        onChange={e => setSensitivePasswordInput(e.target.value)}
        onKeyPress={e => { if (e.key === 'Enter') checkPassword(); }}
      />
      {sensitivePasswordError && <p className="text-sm text-red-600 mt-2">{sensitivePasswordError}</p>}
      <Button onClick={checkPassword} className="mt-4 w-full">Desbloquear</Button>
    </div>
  </div>
  )}
</TabsContent>
  )}
</Tabs>
</CardContent>
</Card>

</div>

<Dialog open={isRequestDialogOpen} onOpenChange={setIsRequestDialogOpen}>
  <DialogContent>
    <DialogHeader><DialogTitle>{editingRequest ? 'Editar' : 'Nova'} Pendência</DialogTitle></DialogHeader>
    <RequestForm
      patientId={patientId}
      requestToEdit={editingRequest}
      onDataAdded={handleDataAdded}
      onClose={() => setIsRequestDialogOpen(false)}
    />
  </DialogContent>

```

```

    </Dialog>
  </div>
);
}

```

## components/exams/ExamUploadDialog.jsx

```

import React, { useState, useCallback, useEffect, useRef } from "react";
import { Exam } from "@entities/Exam";
import { UploadFile } from "@integrations/Core";
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogFooter,
} from "@components/ui/dialog";
import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { Label } from "@components/ui/label";
import { Textarea } from "@components/ui/textarea";
import { Card, CardContent } from "@components/ui/card";
import { Progress } from "@components/ui/progress";
import {
  Upload,
  FileText,
  ImageIcon,
  X,
  Save,
  Camera,
  Plus
} from "lucide-react";
import { Alert, AlertDescription } from "@components/ui/alert";
import { format, parseISO } from 'date-fns';

const initialFormData = {
  exam_type: "",
  exam_date: "",
  requesting_doctor: "",
  observations: ""
};

// Corrigido: onExamCreated renomeado para onSaveSuccess para corresponder ao componente pai
export default function ExamUploadDialog({ isOpen, onClose, patientId, onSaveSuccess, examToEdit }) {
  const [formData, setFormData] = useState(initialFormData);
  const [filesToUpload, setFilesToUpload] = useState([]); // Files selected in the current session for upload
  const [dragActive, setDragActive] = useState(false);
  const [uploadProgress, setUploadProgress] = useState(0);
  const [isSaving, setIsSaving] = useState(false); // Renamed from isUploading
  const [error, setError] = useState("");
  const fileInputRef = useRef(null);

  useEffect(() => {
    if (isOpen) {
      setError("");
      setFilesToUpload([]); // Clear newly selected files on dialog open
      if (fileInputRef.current) {
        fileInputRef.current.value = ""; // Clear file input element
      }
    }
  }, [isOpen]);

  if (examToEdit) {
    setFormData({
      exam_type: examToEdit.exam_type || "",
      exam_date: examToEdit.exam_date ? format(parseISO(examToEdit.exam_date), 'yyyy-MM-dd') : "",
      requesting_doctor: examToEdit.requesting_doctor || "",
      observations: examToEdit.observations || "",
      files: examToEdit.files || [] // Existing files (URLs)
    });
  }

```

```

    });
  } else {
    setFormData({
      ...initialFormData,
      exam_date: format(new Date(), 'yyyy-MM-dd'), // Default to today's date for new exam
    });
  }
}
}, [isOpen, examToEdit]);

const handleDrag = useCallback((e) => {
  e.preventDefault();
  e.stopPropagation();
  if (e.type === "dragenter" || e.type === "dragover") {
    setDragActive(true);
  } else if (e.type === "dragleave") {
    setDragActive(false);
  }
}, []);

const handleDrop = useCallback((e) => {
  e.preventDefault();
  e.stopPropagation();
  setDragActive(false);

  const droppedFiles = Array.from(e.dataTransfer.files).filter(
    file => file.type === "application/pdf" || file.type.startsWith("image/")
  );

  if (droppedFiles.length === 0) {
    setError("Apenas arquivos PDF ou imagens são aceitos");
    return;
  }

  setFilesToUpload(prev => [...prev, ...droppedFiles]);
  setError("");
}, []);

const handleFileChange = (e) => {
  const selectedFiles = Array.from(e.target.files).filter(
    file => file.type === "application/pdf" || file.type.startsWith("image/")
  );

  if (selectedFiles.length === 0) {
    setError("Apenas arquivos PDF ou imagens são aceitos");
    return;
  }

  setFilesToUpload(prev => [...prev, ...selectedFiles]);
  setError("");
};

const removeFileToUpload = (index) => {
  setFilesToUpload(prev => prev.filter((_, i) => i !== index));
};

const removeExistingFile = (index) => {
  setFormData(prev => ({
    ...prev,
    files: prev.files.filter((_, i) => i !== index)
  }));
};

const getFileIcon = (fileNameOrUrl, isExisting = false) => {
  const name = isExisting ? fileNameOrUrl.split("/").pop() : fileNameOrUrl.name;
  if (name.endsWith(".pdf")) {
    return <FileText className="w-5 h-5 text-red-500" />;
  } else if (name.endsWith(".png") || name.endsWith(".jpg") || name.endsWith(".jpeg") || name.endsWith(".gif")) {
    return <ImageIcon className="w-5 h-5 text-blue-500" />;
  }
}

```

```

    return <FileText className="w-5 h-5 text-gray-500" />;
  };

  const getFileName = (fileNameOrUrl, isExisting = false) => {
    return isExisting ? fileNameOrUrl.split("/").pop() : fileNameOrUrl.name;
  };

  const getFileSize = (file, isExisting = false) => {
    return isExisting ? "" : `(${(file.size / 1024 / 1024).toFixed(2)} MB)`;
  };

  const handleSubmit = async (e) => {
    e.preventDefault();

    if (!formData.exam_type || !formData.exam_date) {
      setError("Tipo de exame e data são obrigatórios");
      return;
    }

    let finalFiles = formData.files || [];

    if (finalFiles.length === 0 && filesToUpload.length === 0) {
      setError("Adicione pelo menos um arquivo");
      return;
    }

    setIsSaving(true);
    setError("");
    setUploadProgress(0);

    try {
      if (filesToUpload.length > 0) {
        const uploadPromises = filesToUpload.map(file => UploadFile({ file }));
        const uploadResults = await Promise.all(uploadPromises);
        finalFiles = [...finalFiles, ...uploadResults.map(result => result.file_url)];
      }

      setUploadProgress(50);

      const examDataToSave = {
        patient_id: patientId,
        ...formData,
        files: finalFiles,
      };

      let savedExam;
      if (examToEdit) {
        savedExam = await Exam.update(examToEdit.id, examDataToSave);
      } else {
        savedExam = await Exam.create(examDataToSave);
      }

      setUploadProgress(100);

      onSaveSuccess(savedExam);
      onClose();
    } catch (error) {
      console.error("Erro ao salvar exame:", error);
      setError("Erro ao salvar exame. Tente novamente.");
    } finally {
      setIsSaving(false);
      setUploadProgress(0);
    }
  };

  return (
    <Dialog open={isOpen} onOpenChange={onClose}>
      <DialogContent className="sm:max-w-2xl max-h-[90vh] overflow-y-auto">
        <DialogHeader>
          <DialogTitle className="flex items-center gap-2">

```



```

{examToEdit ? (
  <>
    <Camera className="w-5 h-5 text-blue-600" />
    Editar Exame Médico
  </>
) : (
  <>
    <Upload className="w-5 h-5 text-blue-600" />
    Novo Exame Médico
  </>
)}
</DialogTitle>
</DialogHeader>

<form onSubmit={handleSubmit} className="space-y-6">
  {error && (
    <Alert variant="destructive">
      <AlertDescription>{error}</AlertDescription>
    </Alert>
  )}

  {/* Informações do Exame */}
  <div className="space-y-4">
    <div className="grid grid-cols-2 gap-4">
      <div>
        <Label htmlFor="exam_type">Tipo de Exame *</Label>
        <Input
          id="exam_type"
          value={formData.exam_type}
          onChange={(e) => setFormData(prev => ({...prev, exam_type: e.target.value}))}
          placeholder="Ex: Hemograma, Raio-X"
          required
        />
      </div>
      <div>
        <Label htmlFor="exam_date">Data do Exame *</Label>
        <Input
          id="exam_date"
          type="date"
          value={formData.exam_date}
          onChange={(e) => setFormData(prev => ({...prev, exam_date: e.target.value}))}
          required
        />
      </div>
    </div>

    <div>
      <Label htmlFor="requesting_doctor">Médico Solicitante</Label>
      <Input
        id="requesting_doctor"
        value={formData.requesting_doctor}
        onChange={(e) => setFormData(prev => ({...prev, requesting_doctor: e.target.value}))}
        placeholder="Dr. João Silva"
      />
    </div>

    <div>
      <Label htmlFor="observations">Observações</Label>
      <Textarea
        id="observations"
        value={formData.observations}
        onChange={(e) => setFormData(prev => ({...prev, observations: e.target.value}))}
        placeholder="Resumo dos resultados ou observações importantes..."
        rows={3}
      />
    </div>
  </div>

  {/* Upload de Arquivos */}
  <div className="space-y-4">

```

```
<Label htmlFor="file-input">Arquivos do Exame *</Label>
```

```
{/* Drop Zone */}
<label
  htmlFor="file-input"
  onDragEnter={handleDrag}
  onDragLeave={handleDrag}
  onDragOver={handleDrag}
  onDrop={handleDrop}
  className={`border-2 border-dashed rounded-xl p-6 text-center transition-colors cursor-pointer block ${
    dragActive
      ? "border-blue-400 bg-blue-50"
      : "border-gray-200 hover:border-gray-300"
  }`}
>
  <input
    type="file"
    multiple
    accept=".pdf,.png,.jpg,.jpeg"
    onChange={handleFileChange}
    className="hidden"
    id="file-input"
    ref__={fileInputRef}
  />
```

```
<div className="space-y-3">
  <div className="w-12 h-12 mx-auto bg-gray-100 rounded-full flex items-center justify-center">
    <Plus className="w-6 h-6 text-gray-500" />
  </div>
```

```
<div>
  <p className="font-medium text-gray-900">Solte os arquivos aqui ou clique para selecionar</p>
</div>
```

```
<p className="text-xs text-gray-400">
  PDF, PNG, JPG até 20MB cada
</p>
</div>
</label>
```

```
{/* Lista de Arquivos Existentes */}
{formData.files && formData.files.length > 0 && (
  <div className="space-y-2">
    <p className="font-medium text-sm">Arquivos Existentes:</p>
    {formData.files.map((fileUrl, index) => (
      <Card key={`existing-${index}`} className="border">
        <CardContent className="p-3">
          <div className="flex items-center justify-between">
            <div className="flex items-center gap-3">
              {getFileIcon(fileUrl, true)}
            <div>
              <p className="font-medium text-sm">{getFileName(fileUrl, true)}</p>
              <a href__={fileUrl} target="_blank" rel="noopener noreferrer" className="text-xs text-blue-500 hover:underline">
                Visualizar
              </a>
            </div>
          </div>
          <div>
            <Button
              type="button"
              variant="ghost"
              size="icon"
              onClick={() => removeExistingFile(index)}
            >
              <X className="w-4 h-4" />
            </Button>
          </div>
        </CardContent>
      </Card>
    ))}
  </div>
```

```

    })

    { /* Lista de Novos Arquivos para Upload */ }
    {filesToUpload.length > 0 && (
      <div className="space-y-2">
        <p className="font-medium text-sm">Novos Arquivos para Upload:</p>
        {filesToUpload.map((file, index) => (
          <Card key={`new-${index}`} className="border">
            <CardContent className="p-3">
              <div className="flex items-center justify-between">
                <div className="flex items-center gap-3">
                  {getFileIcon(file)}
                  <div>
                    <p className="font-medium text-sm">{file.name}</p>
                    <p className="text-xs text-gray-500">
                      {(file.size / 1024 / 1024).toFixed(2)} MB
                    </p>
                  </div>
                </div>
                <Button
                  type="button"
                  variant="ghost"
                  size="icon"
                  onClick={() => removeFileToUpload(index)}
                >
                  <X className="w-4 h-4" />
                </Button>
              </div>
            </CardContent>
          </Card>
        ))}
      </div>
    )}
  </div>

  { /* Progress Bar */ }
  {isSaving && (
    <div className="space-y-2">
      <div className="flex justify-between text-sm">
        <span>Salvando exame...</span>
        <span>{uploadProgress}%</span>
      </div>
      <Progress value={uploadProgress} className="h-2" />
    </div>
  )}

  { /* Buttons */ }
  <DialogFooter>
    <Button type="button" variant="outline" onClick={onClose} disabled={isSaving}>
      Cancelar
    </Button>
    <Button
      type="submit"
      disabled={isSaving}
      className="bg-gradient-to-r from-blue-500 to-blue-600 hover:from-blue-600 hover:to-blue-700"
    >
      {isSaving ? (
        <>
          <div className="animate-spin rounded-full h-4 w-4 border-b-2 border-white mr-2"></div>
          Salvando...
        </>
      ) : (
        <>
          <Save className="w-4 h-4 mr-2" />
          {examToEdit ? "Salvar Alterações" : "Salvar Exame"}
        </>
      )}
    </Button>
  </DialogFooter>
</form>

```

```

    </DialogContent>
  </Dialog>
);
}

```

## components/exams/ExamCard.jsx

```

import React from "react";
import { Card, CardContent, CardHeader, CardFooter } from "@components/ui/card";
import { Badge } from "@components/ui/badge";
import { Button } from "@components/ui/button";
import {
  TestTube,
  Calendar,
  User,
  Eye,
  FileText,
  ImageIcon,
  Edit,
  Trash2
} from "lucide-react";
import { format } from "date-fns";
import { ptBR } from "date-fns/locale";

export default function ExamCard({ exam, onView, onEdit, onDelete }) {
  const getStatusColor = (status) => {
    switch (status) {
      case 'completed': return 'bg-green-100 text-green-800 border-green-200';
      case 'pending': return 'bg-yellow-100 text-yellow-800 border-yellow-200';
      case 'reviewed': return 'bg-blue-100 text-blue-800 border-blue-200';
      default: return 'bg-gray-100 text-gray-800 border-gray-200';
    }
  };

  const getStatusText = (status) => {
    switch (status) {
      case 'completed': return 'Concluído';
      case 'pending': return 'Pendente';
      case 'reviewed': return 'Revisado';
      default: return 'Não informado';
    }
  };

  return (
    <Card className="group hover:shadow-lg transition-all duration-300 border-0 shadow-md flex flex-col justify-between">
      <div>
        <CardHeader className="pb-3">
          <div className="flex items-start justify-between">
            <div className="flex items-center gap-2">
              <div className="p-2 bg-blue-100 rounded-lg group-hover:bg-blue-200 transition-colors">
                <TestTube className="w-5 h-5 text-blue-600" />
              </div>
              <div>
                <h3 className="font-semibold text-gray-900">{exam.exam_type}</h3>
                <div className="flex items-center gap-1 text-sm text-gray-500 mt-1">
                  <Calendar className="w-3 h-3" />
                  {format(new Date(exam.exam_date), "dd/MM/yyyy", { locale: ptBR })}
                </div>
              </div>
            </div>
            <Badge className={` ${getStatusColor(exam.status)} border`} >
              {getStatusText(exam.status)}
            </Badge>
          </div>
        </CardHeader>

        <CardContent className="space-y-4">

```

```

    {exam.requesting_doctor && (
      <div className="flex items-center gap-2 text-sm text-gray-600">
        <User className="w-4 h-4" />
        <span>{exam.requesting_doctor}</span>
      </div>
    )}

    {exam.observations && (
      <p className="text-sm text-gray-600 line-clamp-2">
        {exam.observations}
      </p>
    )}

    {exam.files && exam.files.length > 0 && (
      <div className="flex items-center gap-2 text-sm text-gray-500">
        <div className="flex items-center gap-1">
          <FileText className="w-4 h-4" />
          <span>{exam.files.filter(f => f.includes('.pdf')).length}</span>
        </div>
        <div className="flex items-center gap-1">
          <ImageIcon className="w-4 h-4" />
          <span>{exam.files.filter(f => !f.includes('.pdf')).length}</span>
        </div>
      </div>
    )}

    <Button
      variant="outline"
      className="w-full hover:bg-blue-50 hover:text-blue-700 hover:border-blue-200"
      onClick={() => onView(exam)}
    >
      <Eye className="w-4 h-4 mr-2" />
      Visualizar Exame
    </Button>
  </CardContent>
</div>
<CardFooter className="p-4 pt-0 flex gap-2">
  <Button
    variant="outline"
    size="sm"
    className="w-full"
    onClick={() => onEdit(exam)}
  >
    <Edit className="w-4 h-4 mr-2" />
    Editar
  </Button>
  <Button
    variant="destructive"
    size="sm"
    className="w-full"
    onClick={() => onDelete(exam.id)}
  >
    <Trash2 className="w-4 h-4 mr-2" />
    Apagar
  </Button>
</CardFooter>
</Card>
);
}

```

---

## components/exams/ExamViewer.jsx

```

import React from "react";
import {
  Dialog,
  DialogContent,
  DialogHeader,

```

```

    DialogTitle,
  } from "@components/ui/dialog";
import { Button } => "@components/ui/button";
import { Badge } from "@components/ui/badge";
import {
  TestTube,
  Calendar,
  User,
  Download,
  ExternalLink,
  FileText
} from "lucide-react";
import { format } from "date-fns";
import { ptBR } from "date-fns/locale";

export default function ExamViewer({ exam, onClose }) {
  if (!exam) return null;

  const getStatusColor = (status) => {
    switch (status) {
      case 'completed': return 'bg-green-100 text-green-800 border-green-200';
      case 'pending': return 'bg-yellow-100 text-yellow-800 border-yellow-200';
      case 'reviewed': return 'bg-blue-100 text-blue-800 border-blue-200';
      default: return 'bg-gray-100 text-gray-800 border-gray-200';
    }
  };

  const getStatusText = (status) => {
    switch (status) {
      case 'completed': return 'Concluído';
      case 'pending': return 'Pendente';
      case 'reviewed': return 'Revisado';
      default: return 'Não informado';
    }
  };

  return (
    <Dialog open={!exam} onOpenChange={onClose}>
      <DialogContent className="sm:max-w-4xl max-h-[90vh] overflow-y-auto">
        <DialogHeader>
          <DialogTitle className="flex items-center justify-between">
            <div className="flex items-center gap-2">
              <TestTube className="w-5 h-5 text-blue-600" />
              {exam.exam_type}
            </div>
            <Badge className={` ${getStatusColor(exam.status)} border`}>
              {getStatusText(exam.status)}
            </Badge>
          </DialogTitle>
        </DialogHeader>

        <div className="space-y-6">
          {/* Informações do Exame */}
          <div className="grid grid-cols-1 md:grid-cols-2 gap-4 p-4 bg-gray-50 rounded-lg">
            <div className="flex items-center gap-2">
              <Calendar className="w-4 h-4 text-gray-500" />
              <div>
                <span className="text-sm text-gray-500">Data do Exame</span>
                <p className="font-medium">
                  {format(new Date(exam.exam_date), "dd/MM/yyyy", { locale: ptBR })}
                </p>
              </div>
            </div>

            <div>
              <span className="text-sm text-gray-500">Médico Solicitante</span>
              <p className="font-medium">{exam.requesting_doctor}</p>
            </div>
          </div>
        </div>
      </DialogContent>
    </Dialog>
  );
}

```

```

        </div>
      </div>
    })
  </div>

  {/* Observações */}
  {exam.observations && (
    <div>
      <h3 className="font-semibold text-gray-900 mb-2">Observações</h3>
      <div className="p-4 bg-blue-50 rounded-lg">
        <p className="text-gray-700">{exam.observations}</p>
      </div>
    </div>
  )}

  {/* Resumo dos Resultados */}
  {exam.results_summary && (
    <div>
      <h3 className="font-semibold text-gray-900 mb-2">Resumo dos Resultados</h3>
      <div className="p-4 bg-green-50 rounded-lg">
        <p className="text-gray-700">{exam.results_summary}</p>
      </div>
    </div>
  )}

  {/* Arquivos */}
  {exam.files && exam.files.length > 0 && (
    <div>
      <h3 className="font-semibold text-gray-900 mb-3">Arquivos do Exame</h3>
      <div className="space-y-3">
        {exam.files.map((fileUrl, index) => {
          const fileName = fileUrl.split('/').pop() || `Arquivo ${index + 1}`;
          const isPDF = fileUrl.includes('.pdf');

          return (
            <div key={index} className="flex items-center justify-between p-3 border rounded-lg">
              <div className="flex items-center gap-3">
                <FileText className={`w-5 h-5 ${isPDF ? 'text-red-500' : 'text-blue-500'} ` />
                <div>
                  <p className="font-medium">{fileName}</p>
                  <p className="text-sm text-gray-500">
                    {isPDF ? 'Documento PDF' : 'Imagem'}
                  </p>
                </div>
              </div>
            </div>
          );
        })}

        <div className="flex gap-2">
          <Button
            variant="outline"
            size="sm"
            onClick={() => window.open(fileUrl, '_blank')}
          >
            <ExternalLink className="w-4 h-4 mr-1" />
            Ver
          </Button>
          <Button
            variant="outline"
            size="sm"
            onClick={() => {
              const a = document.createElement('a');
              a.href = fileUrl;
              a.download = fileName;
              a.click();
            }}
          >
            <Download className="w-4 h-4 mr-1" />
            Baixar
          </Button>
        </div>
      </div>
    </div>
  )}

```

```

    );
  })
</div>

  { /* PDF Consolidado */
  {exam.pdf_url && (
    <div className="mt-4 p-4 bg-blue-50 rounded-lg">
      <div className="flex items-center justify-between">
        <div>
          <h4 className="font-medium text-gray-900">PDF Consolidado</h4>
          <p className="text-sm text-gray-600">Todos os arquivos em um único documento</p>
        </div>
        <Button
          className="bg-gradient-to-r from-blue-500 to-blue-600 hover:from-blue-600 hover:to-blue-700"
          onClick={() => window.open(exam.pdf_url, '_blank')}
        >
          <FileText className="w-4 h-4 mr-2" />
          Abrir PDF
        </Button>
      </div>
    </div>
  )}
</div>
  )}
</div>
</DialogContent>
</Dialog>
);
}

```

## components/QuickRegister.jsx

```

import React, { useState, useEffect } from "react";
import { Patient } from "@entities/Patient";
import { Exam } from "@entities/Exam";
import { Medication } from "@entities/Medication";
import { Appointment } from "@entities/Appointment";
import { MedicalHistory } from "@entities/MedicalHistory";
import { Incident } from "@entities/Incident";
import { MedicalRequest } from "@entities/MedicalRequest";
import { Credential } from "@entities/Credential";
import { UploadFile } from "@integrations/Core";
import { Dialog, DialogContent, DialogHeader, DialogTitle } from "@components/ui/dialog";
import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { Label } from "@components/ui/label";
import { Textarea } from "@components/ui/textarea";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@components/ui/select";
import { Badge } from "@components/ui/badge";
import {
  TestTube,
  Pill,
  Calendar,
  FileText,
  AlertTriangle,
  ClipboardList,
  KeyRound,
  ArrowLeft,
  ArrowRight,
  Check,
  Camera,
  Upload
} from "lucide-react";
import { format } from "date-fns";
import { ptBR } from "date-fns/locale";

const RECORD_TYPES = [

```



```

{ id: 'exam', label: 'Exame', icon: TestTube, color: 'bg-orange-100 text-orange-700', entity: 'Exam' },
{ id: 'medication', label: 'Medicação', icon: Pill, color: 'bg-red-100 text-red-700', entity: 'Medication' },
{ id: 'appointment', label: 'Consulta', icon: Calendar, color: 'bg-blue-100 text-blue-700', entity: 'Appointment' },
{ id: 'history', label: 'Histórico', icon: FileText, color: 'bg-purple-100 text-purple-700', entity: 'MedicalHistory' },
{ id: 'incident', label: 'Incidente', icon: AlertTriangle, color: 'bg-yellow-100 text-yellow-700', entity: 'Incident' },
{ id: 'request', label: 'Pendência', icon: ClipboardList, color: 'bg-green-100 text-green-700', entity: 'MedicalRequest' },
{ id: 'credential', label: 'Senha', icon: KeyRound, color: 'bg-gray-100 text-gray-700', entity: 'Credential' }
];

```

```

const QUICK_OPTIONS = {
  exam: [
    'Hemograma Completo', 'Raio-X Tórax', 'Ultrassom Abdominal', 'Eletrocardiograma',
    'Ressonância Magnética', 'Tomografia', 'Colonoscopia', 'Endoscopia'
  ],
  medication: [
    'Dipirona 500mg', 'Paracetamol 750mg', 'Amoxicilina 500mg', 'Omeprazol 20mg',
    'Losartana 50mg', 'Metformina 850mg', 'Sinvastatina 20mg', 'AAS 100mg'
  ],
  appointment: [
    'Clínico Geral', 'Cardiologista', 'Dermatologista', 'Ginecologista',
    'Ortopedista', 'Oftalmologista', 'Neurologista', 'Endocrinologista'
  ],
  history: [
    'Cirurgia', 'Internação', 'Vacina', 'Doença Crônica', 'Alergia', 'Fratura'
  ],
  incident: [
    'Queda', 'Corte', 'Queimadura', 'Torção', 'Contusão', 'Reação Alérgica'
  ],
  request: [
    'Marcar Consulta', 'Fazer Exame', 'Comprar Medicamento', 'Renovar Receita', 'Buscar Resultado'
  ],
  credential: [
    'Portal do Plano', 'Laboratório', 'Hospital', 'Clínica', 'Farmácia'
  ]
};

```

```

export default function QuickRegister({ isOpen, onClose, onSuccess }) {
  const [step, setStep] = useState(1);
  const [patients, setPatients] = useState([]);
  const [selectedPatient, setSelectedPatient] = useState("");
  const [selectedType, setSelectedType] = useState("");
  const [formData, setFormData] = useState({
    date: format(new Date(), 'yyyy-MM-dd'),
    description: "",
    customDescription: "",
    useCustom: false
  });
  const [uploadedFiles, setUploadedFiles] = useState([]);
  const [isUploading, setIsUploading] = useState(false);
  const [isSaving, setIsSaving] = useState(false);

```

```

  useEffect(() => {
    if (isOpen) {
      loadPatients();
      resetForm();
    }
  }, [isOpen]);

```

```

  const loadPatients = async () => {
    try {
      const allPatients = await Patient.list();
      const sortedPatients = allPatients.sort((a, b) => {
        const nameA = (a.full_name || "").toLowerCase();
        const nameB = (b.full_name || "").toLowerCase();
        return nameA.localeCompare(nameB);
      });
      setPatients(sortedPatients);
    } catch (error) {
      console.error("Erro ao carregar pacientes:", error);
    }
  }

```

```

};

const resetForm = () => {
  setStep(1);
  setSelectedPatient("");
  setSelectedType("");
  setFormData({
    date: format(new Date(), 'yyyy-MM-dd'),
    description: "",
    customDescription: "",
    useCustom: false
  });
  setUploadedFiles([]);
};

const handleFileUpload = async (e) => {
  const files = Array.from(e.target.files);
  if (files.length === 0) return;

  setIsUploading(true);
  try {
    const uploadPromises = files.map(file => UploadFile({ file }));
    const results = await Promise.all(uploadPromises);
    setUploadedFiles(prev => [...prev, ...results.map(r => r.file_url)]);
  } catch (error) {
    console.error("Erro no upload:", error);
  } finally {
    setIsUploading(false);
  }
};

const getEntityClass = (entityName) => {
  switch(entityName) {
    case 'Exam': return Exam;
    case 'Medication': return Medication;
    case 'Appointment': return Appointment;
    case 'MedicalHistory': return MedicalHistory;
    case 'Incident': return Incident;
    case 'MedicalRequest': return MedicalRequest;
    case 'Credential': return Credential;
    default: return null;
  }
};

const buildDataToSave = () => {
  const patient = patients.find(p => p.id === selectedPatient);
  const typeConfig = RECORD_TYPES.find(t => t.id === selectedType);
  const description = formData.useCustom ? formData.customDescription : formData.description;

  const baseData = {
    patient_id: selectedPatient,
    created_date: new Date().toISOString()
  };

  switch(selectedType) {
    case 'exam':
      return {
        ...baseData,
        exam_type: description,
        exam_date: formData.date,
        files: uploadedFiles,
        status: 'completed'
      };

    case 'medication':
      return {
        ...baseData,
        commercial_name: description,
        dosage: '-- mg',
        frequency: 'Conforme prescrição',

```

```

    start_date: formData.date,
    status: 'active'
  };

  case 'appointment':
    return {
      ...baseData,
      specialty: description,
      doctor: 'Dr(a). --',
      date: new Date(formData.date + 'T10:00:00').toISOString(),
      status: 'completed'
    };

  case 'history':
    return {
      ...baseData,
      type: 'chronic_disease',
      title: description,
      date: formData.date,
      description: 'Registro rápido'
    };

  case 'incident':
    return {
      ...baseData,
      title: description,
      start_date: formData.date,
      description: 'Registro rápido',
      status: 'active'
    };

  case 'request':
    return {
      ...baseData,
      type: 'exam_request',
      title: description,
      request_date: formData.date,
      requesting_doctor: 'Dr(a). --',
      status: 'pending',
      files: uploadedFiles
    };

  case 'credential':
    return {
      ...baseData,
      service_name: description,
      username: "",
      password: "",
      notes: 'Registro rápido'
    };

  default:
    return baseData;
}
};

const handleSave = async () => {
  if (!selectedPatient || !selectedType) return;

  setIsSaving(true);
  try {
    const typeConfig = RECORD_TYPES.find(t => t.id === selectedType);
    const EntityClass = getEntityClass(typeConfig.entity);
    const dataToSave = buildDataToSave();

    await EntityClass.create(dataToSave);

    const patient = patients.find(p => p.id === selectedPatient);
    const description = formData.useCustom ? formData.customDescription : formData.description;

```

```

    if (onSuccess) {
      onSuccess(`${typeConfig.label} "${description}" foi salvo para ${patient.full_name}`);
    }

    onClose();
  } catch (error) {
    console.error("Erro ao salvar:", error);
  } finally {
    setIsSaving(false);
  }
};

const renderStep1 = () => (
  <div className="space-y-6">
    <div className="text-center">
      <h3 className="text-lg font-semibold text-gray-900 mb-2">Selecione o Paciente</h3>
      <p className="text-gray-600">Para quem você quer fazer o registro?</p>
    </div>

    <Select value={selectedPatient} onChange={setSelectedPatient}>
      <SelectTrigger className="w-full">
        <SelectValue placeholder="Escolha um paciente..." />
      </SelectTrigger>
      <SelectContent>
        {patients.map(patient => (
          <SelectItem key={patient.id} value={patient.id}>
            {patient.full_name}
          </SelectItem>
        ))}
      </SelectContent>
    </Select>

    <div className="flex justify-end">
      <Button
        onClick={() => setStep(2)}
        disabled={!selectedPatient}
        className="bg-blue-600 hover:bg-blue-700"
      >
        Próximo <ArrowRight className="w-4 h-4 ml-2" />
      </Button>
    </div>
  </div>
);

const renderStep2 = () => (
  <div className="space-y-6">
    <div className="text-center">
      <h3 className="text-lg font-semibold text-gray-900 mb-2">Tipo de Registro</h3>
      <p className="text-gray-600">O que você quer registrar?</p>
    </div>

    <div className="grid grid-cols-2 gap-3">
      {RECORD_TYPES.map(type => {
        const Icon = type.icon;
        return (
          <Button
            key={type.id}
            variant={selectedType === type.id ? "default" : "outline"}
            onClick={() => setSelectedType(type.id)}
            className={`h-20 flex flex-col gap-2 ${
              selectedType === type.id ? 'bg-blue-600 text-white' : "
            }`}
          >
            <Icon className="w-6 h-6" />
            <span className="text-sm">{type.label}</span>
          </Button>
        );
      })}
    </div>
  </div>
);

```

```

<div className="flex justify-between">
  <Button variant="outline" onClick={() => setStep(1)}>
    <ArrowLeft className="w-4 h-4 mr-2" /> Voltar
  </Button>
  <Button
    onClick={() => setStep(3)}
    disabled={!selectedType}
    className="bg-blue-600 hover:bg-blue-700"
  >
    Próximo <ArrowRight className="w-4 h-4 ml-2" />
  </Button>
</div>
</div>
);

const renderStep3 = () => {
  const typeConfig = RECORD_TYPES.find(t => t.id === selectedType);
  const quickOptions = QUICK_OPTIONS[selectedType] || [];

  return (
    <div className="space-y-6">
      <div className="text-center">
        <h3 className="text-lg font-semibold text-gray-900 mb-2">Detalhes do Registro</h3>
        <Badge className={typeConfig.color}>{typeConfig.label}</Badge>
      </div>

      <div>
        <Label>Data</Label>
        <Input
          type="date"
          value={formData.date}
          onChange={e => setFormData({...formData, date: e.target.value})}
        />
      </div>

      <div>
        <Label>Descrição</Label>
        {!formData.useCustom ? (
          <div className="space-y-3">
            <div className="grid grid-cols-2 gap-2">
              {quickOptions.map(option => (
                <Button
                  key={option}
                  variant={formData.description === option ? "default" : "outline"}
                  onClick={() => setFormData({...formData, description: option})}
                  className="text-sm h-auto py-2"
                >
                  {option}
                </Button>
              ))}
            </div>
            <Button
              variant="ghost"
              onClick={() => setFormData({...formData, useCustom: true})}
              className="w-full text-blue-600"
            >
              + Escrever descrição personalizada
            </Button>
          </div>
        ) : (
          <div className="space-y-2">
            <Textarea
              placeholder="Digite a descrição..."
              value={formData.customDescription}
              onChange={e => setFormData({...formData, customDescription: e.target.value})}
            />
            <Button
              variant="ghost"
              onClick={() => setFormData({...formData, useCustom: false, customDescription: ""})}
              className="text-sm text-gray-600"
            >

```

```

        >
        Voltar às opções rápidas
      </Button>
    </div>
  })
</div>

<div>
  <Label>Anexar Fotos/Documentos (Opcional)</Label>
  <div className="space-y-2">
    <Input
      type="file"
      multiple
      accept="image/*,.pdf"
      onChange={handleFileUpload}
      disabled={isUploading}
    />
    {uploadedFiles.length > 0 && (
      <div className="text-sm text-green-600">
        {uploadedFiles.length} arquivo(s) anexado(s)
      </div>
    )}
    {isUploading && (
      <div className="text-sm text-blue-600">Enviando arquivos...</div>
    )}
  </div>
</div>

<div className="flex justify-between">
  <Button variant="outline" onClick={() => setStep(2)}>
    <ArrowLeft className="w-4 h-4 mr-2" /> Voltar
  </Button>
  <Button
    onClick={handleSave}
    disabled={isSaving || (!formData.description && !formData.customDescription)}
    className="bg-green-600 hover:bg-green-700"
  >
    {isSaving ? 'Salvando...' : (
      <>
        <Check className="w-4 h-4 mr-2" /> Salvar Registro
      </>
    )}
  </Button>
</div>
</div>
);
};

return (
  <Dialog open={isOpen} onOpenChange={onClose}>
    <DialogContent className="max-w-md">
      <DialogHeader>
        <DialogTitle className="flex items-center gap-2">
          ⚡ Registro Rápido
          <Badge variant="outline">Passo {step}/3</Badge>
        </DialogTitle>
      </DialogHeader>

      <div className="py-4">
        {step === 1 && renderStep1()}
        {step === 2 && renderStep2()}
        {step === 3 && renderStep3()}
      </div>
    </DialogContent>
  </Dialog>
);
}

```

---

## layout.js

```
import React, { useState, useEffect } from "react";
import { Link, useLocation } from "react-router-dom";
import { createPageUrl } from "@/utils";
import { User } from "@/entities/User";
import { Patient } from "@/entities/Patient";
import {
  Heart,
  User as UserIcon,
  FileText,
  TestTube,
  Pill,
  Calendar,
  ClipboardList,
  Search,
  Menu,
  X,
  Logout,
  UserPlus,
  KeyRound,
  FileImage,
  Users
} from "lucide-react";
import {
  Sidebar,
  SidebarContent,
  SidebarGroup,
  SidebarGroupContent,
  SidebarGroupLabel,
  SidebarMenu,
  SidebarMenuButton,
  SidebarMenuItem,
  SidebarHeader,
  SidebarFooter,
  SidebarProvider,
  SidebarTrigger,
} from "@components/ui/sidebar";
import { Button } from "@components/ui/button";
import { Avatar, AvatarFallback, AvatarImage } from "@components/ui/avatar";

const navigationItems = [
  {
    title: "Dashboard",
    url: createPageUrl("Dashboard"),
    icon: Heart,
    color: "text-blue-600"
  },
  {
    title: "Busca",
    url: createPageUrl("Search"),
    icon: Search,
    color: "text-gray-600"
  },
];

export default function Layout({ children, currentPageName }) {
  const location = useLocation();
  const [user, setUser] = useState(null);
  const [patients, setPatients] = useState([]);

  useEffect(() => {
    loadUser();
    loadPatients();
  }, []);

  const loadUser = async () => {
    try {
      const currentUser = await User.me();
    }
  }
}
```

```

    setUser(currentUser);
  } catch (error) {
    console.error("Usuário não autenticado");
  }
};

const loadPatients = async () => {
  try {
    const allPatients = await Patient.list('-created_date');
    // Ordenar pacientes alfabeticamente pelo nome completo
    const sortedPatients = allPatients.sort((a, b) => {
      const nameA = (a.full_name || "").toLowerCase();
      const nameB = (b.full_name || "").toLowerCase();
      return nameA.localeCompare(nameB);
    });
    setPatients(sortedPatients);
  } catch (error) {
    console.error("Erro ao carregar pacientes:", error);
  }
};

const handleLogout = async () => {
  await User.logout();
  window.location.reload();
};

const getPatientDisplayName = (fullName) => {
  if (!fullName) return 'Paciente';
  const names = fullName.split(' ');
  if (names.length > 1) {
    return `${names[0]} ${names[names.length - 1]}`;
  }
  return names[0];
};

return (
  <SidebarProvider>
    <div className="min-h-screen flex w-full bg-gradient-to-br from-slate-50 to-blue-50">
      <style>
        {
          :root {
            --medical-blue: #2563EB;
            --medical-green: #059669;
            --medical-gray: #64748B;
            --medical-light: #F8FAFC;
          }

          .medical-gradient {
            background: linear-gradient(135deg, var(--medical-blue) 0%, var(--medical-green) 100%);
          }

          .glass-effect {
            background: rgba(255, 255, 255, 0.95);
            backdrop-filter: blur(10px);
            border: 1px solid rgba(255, 255, 255, 0.2);
          }

          .smooth-transition {
            transition: all 0.3s cubic-bezier(0.4, 0, 0.2, 1);
          }
        }
      </style>

      <Sidebar className="border-r border-white/20 shadow-xl glass-effect">
        <SidebarHeader className="border-b border-gray-100 p-6">
          <div className="flex items-center gap-3">
            <div className="w-12 h-12 medical-gradient rounded-xl flex items-center justify-center shadow-lg">
              <Heart className="w-6 h-6 text-white" />
            </div>
            <div>

```



```

        <h2 className="font-bold text-gray-900 text-lg">Prontuário</h2>
        <p className="text-sm text-gray-500">Médico Pessoal</p>
      </div>
    </div>
  </SidebarHeader>

  <SidebarContent className="p-3">
    <SidebarGroup>
      <SidebarGroupLabel className="text-xs font-semibold text-gray-400 uppercase tracking-wider px-3 py-2">
        Navegação
      </SidebarGroupLabel>
      <SidebarGroupContent>
        <SidebarMenu>
          {navigationItems.map((item) => {
            const isActive = location.pathname === item.url;

            return (
              <SidebarMenuItem key={item.title}>
                <SidebarMenuButton
                  asChild
                  className={`smooth-transition rounded-xl p-3 mb-1 hover:bg-blue-50 hover:text-blue-700 ${
                    isActive ? 'bg-blue-100 text-blue-700 shadow-sm' : ''
                  }`}
                >
                  <Link to={item.url} className="flex items-center gap-3">
                    <item.icon className={`w-5 h-5 ${item.color}`} />
                    <span className="font-medium">{item.title}</span>
                  </Link>
                </SidebarMenuButton>
              </SidebarMenuItem>
            );
          })}
        </SidebarMenu>
      </SidebarGroupContent>
    </SidebarGroup>

    {/* Pacientes Cadastrados - Ordenados Alfabeticamente */}
    {patients.length > 0 && (
      <SidebarGroup>
        <SidebarGroupLabel className="text-xs font-semibold text-gray-400 uppercase tracking-wider px-3 py-2">
          Pacientes
        </SidebarGroupLabel>
        <SidebarGroupContent>
          <SidebarMenu>
            {patients.map((patient) => {
              const isActive = location.pathname.includes('PatientDetails') &&
                location.search.includes(`id=${patient.id}`);

              return (
                <SidebarMenuItem key={patient.id}>
                  <SidebarMenuButton
                    asChild
                    className={`smooth-transition rounded-xl p-3 mb-1 hover:bg-green-50 hover:text-green-700 ${
                      isActive ? 'bg-green-100 text-green-700 shadow-sm' : ''
                    }`}
                  >
                    <Link
                      to={createPageUrl('PatientDetails?id=${patient.id}`)}
                      className="flex items-center gap-3"
                    >
                      <Avatar className="w-6 h-6">
                        <AvatarImage src={patient.photo_url} alt={patient.full_name} />
                        <AvatarFallback className="text-xs bg-green-100 text-green-700">
                          {getPatientDisplayName(patient.full_name).charAt(0).toUpperCase()}
                        </AvatarFallback>
                      </Avatar>
                      <span className="font-medium text-sm">{getPatientDisplayName(patient.full_name)}</span>
                    </Link>
                  </SidebarMenuButton>
                </SidebarMenuItem>
              );
            })}
          </SidebarMenu>
        </SidebarGroupContent>
      </SidebarGroup>
    )}
  )}

```

```

    );
  }}}
</SidebarMenu>
</SidebarGroupContent>
</SidebarGroup>
  })
</SidebarContent>

<SidebarFooter className="border-t border-gray-100 p-4">
  {user && (
    <div className="flex items-center justify-between">
      <div className="flex items-center gap-3">
        <Avatar className="w-10 h-10">
          <AvatarImage src={user.photo_url} alt={user.full_name} />
          <AvatarFallback className="medical-gradient text-white font-semibold">
            {user.full_name?.charAt(0) || 'U'}
          </AvatarFallback>
        </Avatar>
        <div className="flex-1 min-w-0">
          <p className="font-medium text-gray-900 text-sm truncate">
            {user.full_name || 'Usuário'}
          </p>
          <p className="text-xs text-gray-500 truncate">{user.email}</p>
        </div>
      </div>
      <Button
        variant="ghost"
        size="icon"
        onClick={handleLogout}
        className="hover:bg-red-50 hover:text-red-600 smooth-transition"
      >
        <LogOut className="w-4 h-4" />
      </Button>
    </div>
  )}
</SidebarFooter>
</Sidebar>

<main className="flex-1 flex flex-col">
  <header className="bg-white/80 backdrop-blur-sm border-b border-gray-200 px-6 py-4 md:hidden sticky top-0 z-10">
    <div className="flex items-center gap-4">
      <SidebarTrigger className="hover:bg-gray-100 p-2 rounded-lg smooth-transition" />
      <h1 className="text-xl font-semibold text-gray-900">Prontuário Médico</h1>
    </div>
  </header>

  <div className="flex-1 overflow-auto">
    {children}
  </div>
</main>
</div>
</SidebarProvider>
);
}

```