Documentação Técnica

Sistema Tecendo Saúde

Aplicação de Telemedicina para Regiões Remotas da Amazônia

Projeto: Baixo Amazonas e Tapajós

29 de novembro de 2025

Sumário

1 Introdução

O **Tecendo Saúde** é um sistema de saúde digital criado para conectar pacientes de regiões remotas da Amazônia (especificamente o Baixo Amazonas e região do Tapajós) com profissionais de saúde. O principal diferencial deste sistema é que ele **funciona completamente sem internet** para os pacientes, salvando todos os dados no próprio dispositivo (celular ou computador) e sincronizando automaticamente quando houver conexão disponível.

1.1 Problema que o Sistema Resolve

Muitas comunidades amazônicas não têm acesso regular à internet ou ficam a longas distâncias das unidades de saúde. Quando um paciente sente um sintoma (febre, dor, ferida), ele precisa:

- Viajar horas de barco até a cidade mais próxima
- Esperar dias para conseguir atendimento
- Às vezes o problema se agrava pela demora

Com o Tecendo Saúde, o paciente pode:

- **Registrar sintomas** (texto, fotos, vídeos, áudios) diretamente no celular
- **Salvar tudo sem internet** no próprio aparelho
- Quando chegar em uma área com WiFi ou dados móveis, o sistema **envia automaticamente** tudo para os profissionais de saúde
- O profissional responde com orientações, e o paciente vê no celular

1.2 Quem Usa o Sistema

Usuários do Sistema

Pacientes: Moradores das 13 regiões atendidas (Santarém, Belterra, Mojuí dos Campos, etc.)
Profissionais: Agentes Comunitários de Saúde (ACS), enfermeiros e médicos das 9 UBS (Unidades Básicas de Saúde) cadastradas

2 Arquitetura do Sistema

A arquitetura do sistema é dividida em três camadas principais que trabalham juntas:

2.1 Camada de Interface (Frontend)

Esta é a parte que o usuário vê e interage. Foi construída usando:

- **React 18:** Biblioteca JavaScript que cria interfaces dinâmicas. Quando você clica em um botão, o React atualiza apenas a parte necessária da tela (não recarrega tudo).
- **TailwindCSS:** Framework de estilos visuais. Define as cores (verde para saúde), tamanhos de botões, espaçamentos, etc.
- **Arquivo Único (app.html):** Todo o sistema está em um único arquivo HTML de 1.783 linhas. Isso facilita a distribuição: basta copiar este arquivo para qualquer servidor web ou abrir direto no navegador.

Vantagem Técnica

Por usar bibliotecas via CDN (Content Delivery Network), o sistema não precisa de instalação complexa. Não é necessário:

- Node.js ou npm (gerenciadores de pacotes JavaScript)
- Webpack ou ferramentas de build (compiladores)
- Servidor de aplicação complexo

Basta um servidor web simples (Apache, Nginx) ou até abrir o arquivo HTML diretamente.

2.2 Camada de Armazenamento Local (Offline-First)

O coração da funcionalidade offline é o **IndexedDB**, um banco de dados que funciona dentro do navegador. Usamos uma biblioteca chamada **Dexie.js v2** para facilitar o trabalho com ele.

2.2.1 Estrutura de Dados Local

O sistema mantém 3 tabelas no dispositivo do paciente:

1. **perfil:** Dados pessoais do paciente (30+ campos)
   - Nome, CPF, data de nascimento, região, telefone
   - Foto de perfil (armazenada como texto codificado em base64)
   - Histórico médico: hipertensão, diabetes, vícios
   - Metas de saúde: peso ideal, pressão arterial, glicemia
2. **registros:** Consultas/atendimentos criados pelo paciente

- ID único do registro (UUID gerado automaticamente)
- Texto descritivo dos sintomas
- Status (pendente/respondido)
- Array de interações (até 5 trocas de mensagens)
- Timestamps (data/hora de criação e última atualização)
- Campo `synced`: 0 = precisa enviar, 1 = já enviado

3. **midias:** Arquivos anexados (fotos, vídeos, áudios)
    - Blob (dados binários do arquivo)
    - Tipo MIME (image/jpeg, video/mp4, audio/webm)
    - Nome do arquivo
    - Referência ao registro (qual consulta pertence)
    - Campo `synced` para controlar envio

2.3 Camada de Sincronização (Motor Automático)

O sistema possui um **motor de sincronização** que executa continuamente em segundo plano:

```
// Pseudo-codigo do motor de sincronizacao
setInterval(async () => {
  if (!navigator.onLine) return; // Se offline, aguarda

  // 1. Buscar perfis nao enviados (synced=0)
  const perfis = await db.perfil.where('synced').equals(0);
  for (const perfil of perfis) {
    await supabase.upsert(perfil); // Enviar para nuvem
    await db.perfil.update(perfil.id, { synced: 1 }); // Marcar como enviado
  }

  // 2. Buscar registros nao enviados
  // 3. Buscar midias nao enviadas
  // ... (mesmo processo)
}, 5000); // Executar a cada 5 segundos
```

Como Funciona na Prática

Cenário 1 - Paciente sem internet:

1. Paciente abre o app e cria registro: *"Estou com febre há 3 dias"*
2. Tira foto do termômetro mostrando 38.5°C
3. Clica em ENVIAR
4. Sistema salva TUDO localmente (IndexedDB)
5. Motor de sync detecta `synced=0` mas não há internet → aguarda

Cenário 2 - Paciente volta à cidade (3 dias depois):

1. Celular detecta WiFi automaticamente
2. Motor de sync verifica: "Há 1 registro + 1 foto com `synced=0`"
3. Envia para o Supabase (backend na nuvem)
4. Marca como `synced=1`
5. Badge visual desaparece: "Sincronizando 2 itens..."→ "Tudo enviado"

2.4 Camada de Backend (Supabase)

O backend utiliza **Supabase**, uma plataforma serverless que combina:

- **PostgreSQL:** Banco de dados relacional para armazenar dados estruturados
- **Storage:** Armazenamento de arquivos (fotos, vídeos, áudios)

- **API REST automática:** Gerada automaticamente a partir das tabelas

2.4.1 Tabelas no Supabase

1. **perfis:** Mesma estrutura da tabela local (30+ colunas)
   - Chave única: `patient_id` (identificador do paciente)
   - CPF armazenado sem máscara (apenas números)
   - Timestamps automáticos: `created_at`, `updated_at`
2. **registros:** Consultas sincronizadas
   - Chave única: `registro_id` (UUID gerado localmente)
   - Campo especial: `replies_json` (JSONB - formato flexível)
   - Este campo armazena o histórico completo de interações como array JSON
3. **profissionais:** Dados dos ACS, enfermeiros, médicos
   - Login via CPF (sem senha por questões de acessibilidade)
   - Nome (escolhido de lista de 92 agentes cadastrados)
   - UBS de atuação (9 unidades disponíveis)

2.4.2 Bucket de Storage

Estrutura de pastas no Supabase Storage (forma textual, sem caracteres especiais):

```
midias/  (bucket publico)
  <UUID do registro>/
    foto_<timestamp>.jpg
    video_<timestamp>.mp4
    audio_<timestamp>.webm
```

Segurança e Privacidade

Por que usar UUID em vez de CPF na pasta?
Se usássemos CPF como nome da pasta (por exemplo, `midias/12345678900/foto.jpg`), qualquer pessoa que soubesse o CPF poderia acessar as fotos do paciente apenas digitando a URL. O UUID é um código aleatório de 36 caracteres que não pode ser adivinhado, garantindo que apenas quem tem o link exato pode acessar.
Trade-off: O sistema não usa autenticação por senha (RLS desabilitado) porque:

- Pacientes em áreas remotas não têm email confiável
- SMS tem custo alto e cobertura limitada
- CPF único é suficiente para identificação básica

3 Fluxos de Uso Detalhados

3.1 Fluxo do Paciente (100% Offline)

3.1.1 Primeiro Acesso - Cadastro Básico

1. **Tela Inicial:** Paciente abre o app e clica em "SOU PACIENTE"
2. **Tela de Login:** Sistema solicita CPF
   - Máscara automática: `000.000.000-00`
   - Input numérico (facilita digitação no celular)
3. **Verificação:** Sistema busca CPF em 2 lugares (nesta ordem):
   a) IndexedDB local (prioridade - instantâneo)
   b) Supabase online (se houver conexão - 2–3 segundos)
4. **CPF não encontrado:** Sistema redireciona para cadastro básico
   - CPF já vem preenchido automaticamente
   - 6 campos obrigatórios:
     - Nome completo (texto livre)
     - Data de nascimento (máscara DD/MM/AAAA)
     - Região (dropdown com 13 municípios)
     - Telefone (máscara automática com DDD)
     - Email (opcional)
5. **Salvamento:**

```
const basicProfile = {
  nome: "Joao da Silva",
  cpf: "12345678900", // Sem mascara
  nascimento: "15/03/1985",
  regiao: "Santarem",
  telefone: "(91) 98765-4321",
  email: "joao@exemplo.com",
  patientId: "cpf-12345678900", // ID unico
  synced: 0 // Marca para sincronizar depois
};
await db.perfil.add(basicProfile);
localStorage.setItem('currentProfileId', 'cpf-12345678900');
```

6. **Dashboard:** Paciente entra automaticamente (sem senha)

3.1.2 Criando um Novo Registro

1. **Dashboard:** Paciente clica no botão "NOVO REGISTRO"
2. **Tela de Criação:**
   - Campo de texto grande: "Descreva o que você está sentindo..."

- 3 botões de mídia:
  - Foto (abre câmera do dispositivo)
  - Vídeo (grava vídeo curto)
  - Gravar (captura áudio via microfone)

3. **Anexando Múltiplas Mídias:**
   - Paciente pode anexar vários arquivos
   - Sistema armazena em array: `files = [foto1, video1, audio1]`
   - Preview visual antes de enviar
   - Botão "Remover"para cada arquivo
4. **Envio e Salvamento:**

```
// 1. Gerar UUID unico para este registro
const registroId = uuidv4();

// 2. Salvar registro na tabela
await db.registros.add({
  registroId: registroId,
  patientId: "cpf-12345678900",
  texto: "Estou com febre ha 3 dias",
  tipo: "foto", // Tipo do primeiro arquivo
  status: "pendente",
  createdAt: new Date().toISOString(),
  synced: 0 // Precisa sincronizar
});

// 3. Salvar todos os arquivos anexados
for (const file of files) {
  await db.midias.add({
    registroId: registroId,
    name: file.name,
    type: file.type, // "image/jpeg"
    blob: file,
    synced: 0
  });
}
```

5. **Feedback Visual:**
   - Alerta: "Salvo!"
   - Retorna ao dashboard
   - Badge flutuante aparece: "Sincronizando itens..."

3.1.3 Visualizando Histórico e Respostas

1. **Dashboard - Histórico:**
   - Lista de cards (um por registro)

- Cada card mostra:
  - Data/hora (formato brasileiro)
  - Texto resumo
  - Thumbnail da primeira mídia
  - Badge de status (pendente/respondido)
- Atualização automática a cada 15 segundos (se online)

2. **Modal de Detalhes:** Ao clicar no card:
   - Mostra texto completo
   - Todas as mídias anexadas
   - Vídeos inline
   - Fotos clicáveis
   - Áudios com player
   - Histórico de interações (até 5 trocas)

3.2 Fluxo do Profissional (Requer Internet)

3.2.1 Login e Cadastro

1. **Tela Inicial:** Profissional clica "ÁREA PROFISSIONAL"
2. **Login via CPF:**
   - Sistema busca no Supabase (requer internet)
   - Query SQL executada:

```sql
SELECT * FROM profissionais
WHERE cpf = '12345678900'
LIMIT 1;
```

3. **CPF não encontrado:** Cadastro rápido
   - CPF preenchido automaticamente
   - Nome: dropdown com 92 ACS cadastrados
   - UBS: dropdown com 9 unidades
   - Telefone: máscara automática

3.2.2 Contadores Automáticos

```javascript
// Loop 1: Novos Cadastros (executa a cada 30 segundos)
setInterval(async () => {
  const { data } = await supabase
    .from('perfis')
    .select('cpf, nome, foto_url, genero')
    .not('nome', 'is', null)
    .not('cpf', 'is', null);

  // Filtrar pacientes com dados basicos mas sem prontuario completo
  const incomplete = data.filter(p =>
    !p.foto_url || !p.genero
  );

  setNewPatientCount(incomplete.length);
}, 30000);

// Loop 2: Atendimentos Pendentes
setInterval(async () => {
  const { data } = await supabase
    .from('registros')
    .select('status');

  const pending = data.filter(r =>
    r.status === 'pendente' || r.status === 'em_analise'
  );

  setPendingCount(pending.length);
}, 30000);
```

3.2.3 Painel de Atendimentos

```
-- 1. Buscar todos os registros
SELECT * FROM registros
ORDER BY updated_at DESC;

-- 2. Buscar perfis completos
SELECT * FROM perfis;

-- 3. Para cada registro, buscar primeira midia
-- (feito via API de storage do Supabase)
```

3.2.4 Chat com Paciente

```
// Busca em 2 fontes (nesta ordem):
// 1. IndexedDB local (se profissional usou offline)
const localMedias = await db.midias
  .where('registroId').equals(registroId)
  .toArray();

// 2. Supabase Storage (se online)
const { data: files } = await supabase.storage
  .from('midias')
  .list(registroId);

// Para cada arquivo, detectar tipo
let type;
if (file.name.includes('.mp4')) type = 'video';
else if (file.name.includes('.jpg')) type = 'image';
else if (file.name.includes('.mp3')) type = 'audio';
```

```
// Exemplo de renderizacao de midias (pseudo-React)
<audio controls src={url} className="w-full" />

<video controls src={url}
       className="w-full rounded border" />

<img src={url}
     onClick={() => abrirModal(url, 'image')}
     className="w-24 h-24 cursor-pointer" />
```

```
UPDATE registros SET
  status = 'respondido',
  updated_at = NOW(),
  replies_json = replies_json ||
    '[{"from":"pro", "text":"...", "urls":[...]}]',
  resposta = 'Texto resumo da ultima resposta'
WHERE id = registro_id;
```

3.2.5 Ficha Médica Completa

```
const REQUIRED_KEYS = [
  'nome', 'cpf', 'nascimento', 'genero', 'raca',
  'endereco', 'telefone', 'escolaridade', 'profissao',
  'mora_sozinho', 'regiao', 'ubs_referencia',
  'acs_responsavel', 'hipertensao', 'diabetes',
  'vicios', 'tempo_vicio', 'peso_inicial',
  'enxerga_bem', 'consulta_oftalmo',
  'uso_medicacoes', 'atividade_fisica',
  'freq_atividade', 'tipo_atividade',
  'meta_peso', 'meta_glicemia',
  'meta_pa_min', 'meta_pa_max'
];

for (const key of REQUIRED_KEYS) {
  if (!form[key]) {
    alert('Preencha todos os campos obrigatorios.');
    return;
  }
}
```

```
INSERT INTO perfis (
  patient_id, nome, cpf, nascimento, foto_url,
  genero, raca, endereco, telefone
) VALUES (
  'cpf-12345678900', 'Joao da Silva',
  '12345678900', '15/03/1985', '...',
  'Masculino', 'Pardo', 'Rua das Mangueiras...', '...'
)
ON CONFLICT (patient_id) DO UPDATE SET
  nome = EXCLUDED.nome,
  foto_url = EXCLUDED.foto_url,
  updated_at = NOW();
```

4 Detalhes Técnicos Avançados

4.1 Estrutura do Campo replies_json (JSONB)

```
[
  {
    "from": "paciente",
    "text": "Estou com febre ha 3 dias",
    "url": "https://supabase.co/.../foto1.jpg",
    "urls": [
      "https://.../foto1.jpg",
      "https://.../video1.mp4"
    ],
    "at": "2025-11-29T10:30:00.000Z"
  },
  {
    "from": "pro",
    "pro_name": "Dra. Maria Silva",
    "pro_ubs": "UBS Boa Esperanca",
    "pro_cpf": "12345678900",
    "text": "Recomendo paracetamol 500mg",
    "url": "https://.../resposta_audio.webm",
    "urls": [
      "https://.../resposta_audio.webm",
      "https://.../receita_foto.jpg"
    ],
    "at": "2025-11-29T14:45:00.000Z"
  }
]
```

4.2 Geração de UUIDs

```
function uuidv4() {
  return ([1e7]+-1e3+-4e3+-8e3+-1e11)
    .replace(/[018]/g, c =>
      (c ^ crypto.getRandomValues(new Uint8Array(1))[0]
       & 15 >> c / 4).toString(16)
    );
}
// Exemplo: "c5e95ee9-f9ca-46e1-9026-8e27da42605f"
```

4.3 Conversão de Arquivos para Base64

```
const fileToDataUrl = (file) => new Promise((resolve, reject) => {
  const reader = new FileReader();
  reader.onload = () => resolve(reader.result);
  reader.onerror = reject;
  reader.readAsDataURL(file);
});
```

```
// Uso:
const fotoBase64 = await fileToDataUrl(fotoFile);
// "..."
```

4.4 Máscaras de Input

```
// CPF: 000.000.000-00
const mascaraCPF = (v) =>
  v.replace(/\D/g,'')
   .replace(/(\d{3})(\d)/,'$1.$2')
   .replace(/(\d{3})(\d)/,'$1.$2')
   .replace(/(\d{3})(\d{1,2})/,'$1-$2')
   .replace(/(-\d{2})\d+?$/,'$1');

// Telefone
const mascaraTelefone = (v) => {
  const nums = v.replace(/\D/g,'');
  if(nums.length <= 10)
    return nums.replace(/(\d{2})(\d)/,'($1) $2')
               .replace(/(\d{4})(\d)/,'$1-$2');
  return nums.replace(/(\d{2})(\d)/,'($1) $2')
             .replace(/(\d{5})(\d)/,'$1-$2');
};

// Data: DD/MM/AAAA
const mascaraData = (v) =>
  v.replace(/\D/g,'')
   .replace(/(\d{2})(\d)/,'$1/$2')
   .replace(/(\d{2})(\d)/,'$1/$2')
   .replace(/(\d{4})\d+?$/,'$1');
```

4.5 Cálculo Automático de Idade

```
const calcAge = (dataNascimento) => {
  if (!dataNascimento) return '';
  const [dia, mes, ano] = dataNascimento.split('/');
  const nascimento = new Date(`${ano}-${mes}-${dia}`);
  const diff = Date.now() - nascimento.getTime();
  const anos = Math.floor(diff / 31557600000);
  return `${anos} anos`;
};
```

4.6 Formatação de Timestamps

```
function formatDateISO(iso) {
  try {
    return new Date(iso).toLocaleString('pt-BR');
```

```
  } catch(e) {
    return iso || '';
  }
}
```

5 Limitações e Considerações

5.1 Limitações do Plano Gratuito do Supabase

Recurso	**Gratuito**	**Pro ($25/mês)**
Storage	1 GB	100 GB
Bandwidth	2 GB/mês	200 GB/mês
PostgreSQL	500 MB	8 GB

Tabela 1: Limites do Supabase

5.2 Limitações de Segurança

Login via CPF sem senha, RLS desabilitado e anon key exposta no código são decisões conscientes para atender o contexto de regiões remotas, com as devidas restrições e cuidados.

5.3 Limitações de Escalabilidade

Arquitetura em arquivo único HTML, limites do IndexedDB e possíveis falhas silenciosas de sincronização são pontos a monitorar em produção.

6 Conclusão

O sistema **Tecendo Saúde** demonstra que é possível criar aplicações de saúde digital funcionais para regiões com infraestrutura precária usando tecnologias web modernas. As principais conquistas técnicas foram:

1. **Offline-first real**
2. **Arquitetura simples**
3. **Suporte a múltiplas mídias**
4. **Prontuário eletrônico completo**
5. **Chat assíncrono funcional**

6.1 Trabalhos Futuros

1. Notificações push
2. Exportação de PDF
3. Integração e-SUS
4. Gráficos de evolução
5. Videochamada WebRTC
6. Modo escuro