

# Guia de Continuidade - Frontend SAÚDE!

Este documento serve como um guia abrangente para a nova equipe que assumirá o desenvolvimento do projeto SAÚDE!, um aplicativo móvel e sistema web focado em facilitar o acompanhamento de pessoas com sofrimento psíquico.

## Visão Geral

O SAÚDE! é um sistema em constante evolução que busca integrar pacientes, Agentes Comunitários de Saúde (ACS), psicólogos e psiquiatras de forma humanizada e extensível.

Suas principais funcionalidades incluem:

- Registro de hábitos, sintomas e sentimentos de pacientes
- Personalização do acompanhamento
- Compartilhamento de dados com profissionais mediante autorização do paciente
- Utilização do modelo de dados OMOP para garantir extensibilidade e interoperabilidade
- Desenvolvimento com foco na RAPS (Rede de Atenção Psicossocial)

## Estrutura do Projeto

O projeto é composto por duas partes principais: o frontend (aplicativo mobile e sistema web) e o backend.

## Frontend

O frontend é desenvolvido em React e visa proporcionar uma experiência fluida tanto em dispositivos móveis quanto na web.

## Backend

O backend é construído com Django, responsável pela autenticação, gerenciamento de banco de dados e outras funcionalidades essenciais.

## Instalação e Execução do Frontend

Para começar a trabalhar com o frontend, siga os passos abaixo:

1. Clone o repositório:  
`git clone <git@github.com>:datasci4citizens/app-saude.git`
2. Navegue até o diretório do projeto:  
`cd app_saude`
3. Copie o arquivo de modelo de variáveis de ambiente:  
`cp .env.model .env`
4. Instale as dependências:  
`npm install`
5. Inicie o servidor de desenvolvimento:  
`npm run dev`

## Estrutura de Pastas

O projeto SAÚDE! apresenta uma arquitetura moderna e bem estruturada, característica de uma aplicação híbrida que atende simultaneamente plataformas web e móveis. A base do projeto é construída em React com TypeScript, aproveitando o framework Capacitor para converter a aplicação web em apps nativos para Android e iOS. Analisando sua estrutura, observamos uma organização cuidadosa que facilita tanto o desenvolvimento quanto a manutenção.

O coração do código encontra-se na pasta `src/`, que abriga todo o código React/TypeScript compartilhado entre as diferentes plataformas, seguindo o princípio de "escreva uma vez, execute em qualquer lugar". Complementarmente, as pastas `android/` e `ios/` contêm as configurações específicas para cada

plataforma móvel, incluindo recursos nativos como ícones, telas de splash e arquivos de manifesto necessários para a distribuição nas lojas de aplicativos.

A organização interna do código-fonte segue padrões modernos de desenvolvimento React, com uma clara separação entre componentes reutilizáveis (**components/**), páginas completas (**pages/**), serviços de API (**api/**) e utilitários (**utils/**). Particularmente notável é a rica biblioteca de componentes UI, dividida logicamente entre componentes de formulário e elementos de interface geral, promovendo consistência visual e facilitando a manutenção. A estrutura de páginas revela uma importante característica do aplicativo: a separação clara entre funcionalidades para pacientes (**patient/**) e para profissionais de saúde (**provider/**), indicando que o sistema atende diferentes perfis de usuários com necessidades específicas - pacientes têm acesso a diários, registro de emergência e lembretes, enquanto profissionais podem visualizar pacientes, acessar relatórios e responder a emergências.

A integração com backend é gerenciada através de código gerado automaticamente na pasta **api/**, com modelos TypeScript fortemente tipados que refletem a estrutura de dados do backend Django REST Framework. A presença de numerosos modelos como **ConceptCreate.ts** e **DomainCreate.ts**, junto com o utilitário **conceptLoader.tsx**, sugere a implementação do padrão OMOP (Observational Medical Outcomes Partnership), um modelo padronizado para dados de saúde que visa interoperabilidade e extensibilidade.

O desenvolvimento é suportado por uma infraestrutura moderna com automação de CI/CD via GitHub Actions, hooks de git com Husky para garantia de qualidade de código, containerização com Docker para ambientes de desenvolvimento consistentes, e scripts automatizados para geração de código de API.


Textproto


### **.github/**

- workflows/

- main.yml  Configuração de CI/CD para integração e deploy contínuo

### **.husky/**

- pre-commit  Script que roda antes de cada commit para garantir qualidade de código

### **android/** 

> Configuração e recursos para o app Android gerado pelo Capacitor

- app/ 📱
  - release/ 📦
    - output-metadata.json 📄 Metadados da build de release
  - src/ 🌿
    - main/ 🔧
      - java/ ☕
        - br/unicamp/ic/boilerplate/
          - MainActivity.java 📱 Atividade principal do app Android
      - res/ 🖼️
        - drawable/ 🎨 Imagens e recursos visuais
        - valores de densidades (hdpi, mdpi, etc) 🖼️ Recursos para diferentes tamanhos de tela
        - values/ 📄 Strings e estilos
        - xml/ 📄 Arquivos de configuração XML
          - AndroidManifest.xml 📄 Configuração principal do app Android
    - build.gradle 🔧 Script de build para o Android
    - capacitor.build.gradle ⚡ Configurações do Capacitor para Android

### ### icons/ 🖼️

> Ícones do aplicativo em vários tamanhos

- icon-128.webp, icon-192.webp, etc. 🖼️ Ícones em diferentes tamanhos

### ### ios/ 🍏

> Configuração e recursos para o app iOS gerado pelo Capacitor

- App/ 📱 Projeto iOS
  - Assets.xcassets/ 🖼️ Recursos visuais iOS
  - Base.lproj/ 📄 Arquivos de storyboard e interface
  - AppDelegate.swift 📱 Código principal do app iOS
  - Info.plist 📄 Configurações do app iOS

### ### resources/ 🖼️

- icon.png 🖼️ Ícone principal do aplicativo
- splash.png 🖼️ Tela de splash do aplicativo

### ### src/ 💻

> Código fonte principal da aplicação React

### #### api/ ⚡

> Código gerado automaticamente para comunicação com o backend

























- core/ 🔧 Utilitários base para API
  - ApiError.ts ❌ Gerenciamento de erros da API
  - ApiRequestOptions.ts 🔧 Opções para requisições
  - ApiResult.ts 📊 Respostas padronizadas
  - CancelablePromise.ts ⛔ Promise com capacidade de cancelamento

- OpenAPI.ts 🔑 Configurações principais da API
- request.ts 📡 Função para fazer requisições
- models/ 📊
  - Diversos arquivos .ts 📝 Interfaces TypeScript que representam os modelos de dados do backend
- services/ 🛠️
  - AccountService.ts 👤 Serviço para gerenciamento de contas
  - AuthService.ts 🔑 Serviço para autenticação
  - Diversos outros serviços 🔄 Serviços para diferentes recursos da API


#### #### components/ 🧩

> Componentes React reutilizáveis




- forms/ 📝
  - button.tsx ⬤ Componente de botão
  - checkbox.tsx ✅ Componente de checkbox
  - date\_input.tsx 📅 Input para seleção de data
  - dropdown-menu.tsx ▼ Menu suspenso
  - form.tsx 📄 Componente base para formulários
  - input.tsx 📄 Campo de entrada de texto
  - multi\_select\_custom.tsx 📄 Seletor múltiplo customizado
  - progress\_indicator.tsx 📊 Indicador de progresso
  - radio-checkbox.tsx ⬤ Botões de opção
  - radio-group.tsx ⬤ Grupo de botões de opção
  - select\_input.tsx ▼ Campo de seleção
  - text\_input.tsx 📝 Campo de texto
  - wheel-picker.tsx 🎡 Seletor tipo roda
- ui/ 🎨
  - accordion.tsx ▼ Componente expansível
  - alert-dialog.tsx ⚠️ Diálogo de alerta
  - alert.tsx ⚠️ Componente de alerta
  - aux\_icons.tsx 🖼️ Ícones auxiliares
  - avatar.tsx 👤 Componente de avatar
  - back\_arrow.tsx ⬅️ Botão de voltar
  - badge.tsx 🏷️ Emblema/etiqueta
  - bottom-sheet.tsx 📄 Folha deslizante inferior
  - calendar.tsx 📅 Calendário
  - card.tsx 📇 Cartão para mostrar informações
  - carousel.tsx 🎠 Carrossel de imagens
  - confirmDialog.tsx ✓ Diálogo de confirmação
  - ContinueButton.tsx ➡️ Botão de continuação
  - dialog.tsx 💬 Janela de diálogo
  - drawer.tsx 🗄️ Gaveta lateral
  - error-message.tsx ❌ Mensagem de erro
  - file-uploader.tsx 📁 Uploader de arquivos

- google-signin.tsx  Botão de login com Google
- habit-card.tsx  Cartão para mostrar hábitos
- header.tsx  Cabeçalho da aplicação
- home-banner.tsx  Banner da página inicial
- icon-button.tsx  Botão com ícone
- info-card.tsx  Cartão de informação
- interests-selector.tsx  Seletor de interesses
- label.tsx  Etiqueta para campos
- labeled-switch.tsx  Interruptor com texto
- navigator-bar.tsx  Barra de navegação
- patient-button.tsx  Botão específico para pacientes
- popover.tsx  Popover de informações
- profile-banner.tsx  Banner do perfil
- ProgressIndicator.tsx  Indicador de progresso
- reminder-card.tsx  Cartão de lembrete
- select\_habit.tsx  Seleccionador de hábitos
- select.tsx  Componente de seleção
- selectable-button.tsx  Botão selecionável
- skeleton.tsx  Placeholder de carregamento
- slider.tsx  Controle deslizante
- success-message.tsx  Mensagem de sucesso
- switch.tsx  Interruptor toggle
- tabs.tsx  Abas para navegação
- task-bar.tsx  Barra de tarefas
- text\_input\_diary.tsx  Campo de texto para diário
- textarea.tsx  Área de texto multilinhas
- ViewButton.tsx  Botão para visualizar conteúdo
- EditInterestsDialog.tsx  Diálogo para editar interesses
- ProtectedRoute.tsx  Componente de rota protegida
- SplashScreen.tsx  Tela de splash da aplicação

#### #### contexts/




- AppContext.tsx  Contexto principal da aplicação

#### #### lib/

- images/ 
  - 2.png, error.png, etc.  Imagens utilizadas na aplicação
- utils.ts  Funções utilitárias gerais

#### #### pages/

> Páginas/telas da aplicação

- landing/ 
  - AccountManager.tsx  Gerenciador de contas
  - AccountSelectionScreen.tsx  Tela de seleção de conta

- ConfirmLogoutScreen.tsx 🚪 Confirmação de logout
- EntryOption.tsx 📄 Opções de entrada
- loginScreen.css 🎨 Estilos da tela de login
- LoginScreen.tsx 🔑 Tela de login
- OnboardingSlider.tsx 🔄 Slider para introdução do app
- Terms.tsx 📄 Tela de termos de uso
- TermsText.tsx 📄 Texto dos termos de uso
- patient/ 👤
  - diary/ 📅
    - Diary.tsx 📝 Diário do paciente
    - DiaryInfoForm.tsx 📄 Formulário de informações do diário
    - DiaryListPage.tsx 📄 Lista de entradas do diário
    - modify-habits.tsx 🛠️ Modificação de hábitos
    - ViewDiary.tsx 👁️ Visualização do diário
  - emergency/ 🚨
    - EmergencyUser.tsx 🆘 Tela de emergência para usuário
  - interests/ 🎯
    - CustomInterestPage.tsx 🎯 Página de interesses personalizados
    - InterestsPage.tsx 🎯 Página de seleção de interesses
  - onboarding/ 🚀
    - UserInfoForm.tsx 📄 Formulário de informações do usuário (passo 1)
    - UserInfoForm2.tsx 📄 Formulário de informações do usuário (passo 2)
    - UserInfoForm3.tsx 📄 Formulário de informações do usuário (passo 3)
    - UserOnBoarding.tsx 🚀 Fluxo de onboarding do usuário
  - profile/ 👤
    - ManageProfessionals.tsx 👤 Gerenciamento de profissionais vinculados
    - ProfilePage.tsx 👤 Página de perfil do paciente
  - reminders/ 🔔
    - NewReminder.tsx ➕ Criar novo lembrete
    - Reminders.tsx 🔔 Lista de lembretes
    - ViewReminder.tsx 👁️ Visualizar lembrete
  - UserMainPage.tsx 🏠 Página principal do usuário paciente
  - ViewSelectedInterests.tsx 👁️ Visualizar interesses selecionados
- provider/ 👤
  - profile/ 👤
    - AcsProfilePage.tsx 👤 Página de perfil do ACS
  - AcsMainPage.tsx 🏠 Página principal do ACS
  - EmergencyPage.tsx 🚨 Página de emergência
  - PatientsPage.tsx 👥 Lista de pacientes
  - ProfessionalInfoForm.tsx 📄 Formulário para profissional
  - ProfessionalOnBoarding.tsx 🚀 Onboarding do profissional
  - ReportsPage.tsx 📊 Página de relatórios
  - ViewDiary.tsx 👁️ Visualização do diário do paciente
  - ViewHelp.tsx 🆘 Visualização de pedidos de ajuda

- ViewPatient.tsx 👁 Visualização detalhada do paciente
- ComponentCatalog.tsx 🌱 Catálogo de componentes (para dev)
- NotFound.tsx 🔍 Página de erro 404
- SelectRegister.tsx 📝 Seleção de tipo de registro

#### #### utils/ 🛠

- conceptLoader.tsx 🔄 Carregador de conceitos do modelo OMOP

#### #### arquivos principais

- App.tsx 🧩 Componente principal da aplicação
- globals.css 🎨 Estilos globais
- main.tsx 🚀 Ponto de entrada da aplicação
- vite-env.d.ts 📝 Tipos para o ambiente Vite

#### ### Arquivos de configuração raiz

- .editorconfig 📏 Configurações do editor
- .env.model 🔑 Modelo de arquivo de variáveis de ambiente
- .gitattributes 📄 Atributos para o Git
- .gitignore 🗑 Arquivos a serem ignorados pelo Git
- .prettierrc 🎨 Configuração do formatador Prettier
- account-deletion.html 🗑 Página de deleção de conta
- biome.json 🌿 Configuração do linter/formatter Biome
- capacitor.config.ts ⚡ Configuração do Capacitor
- COMPONENTS.md 📖 Documentação dos componentes
- design-tokens.tokens.json 🎨 Tokens de design do sistema
- docker-compose.yml 🐳 Configuração Docker para desenvolvimento
- export.json 🍰 Arquivo de exportação
- generate-api.js 🔄 Script para gerar código da API
- image.png 🖼 Imagem do projeto
- index.html 📄 Página HTML principal
- package-lock.json 📦 Controle preciso de versões de dependências
- package.json 📦 Dependências e scripts do projeto
- postcss.config.js 🎨 Configuração do PostCSS
- privacy-policy.html 🔒 Política de privacidade
- README.md 📖 Documentação principal do projeto
- tailwind.config.ts 🎨 Configuração do Tailwind CSS
- TAILWIND.md 📖 Documentação sobre o uso do Tailwind
- tsconfig.app.json ⚙ Configuração TypeScript para app
- tsconfig.app.tsbuildinfo 🏗 Informações de build TypeScript
- tsconfig.json ⚙ Configuração principal do TypeScript
- tsconfig.node.json ⚙ Configuração TypeScript para Node
- tsconfig.node.tsbuildinfo 🏗 Informações de build para Node
- vite-env.d.ts 📝 Declarações de tipos para Vite
- vite.config.ts ⚙ Configuração do bundler Vite



# Uso do tailwind

O projeto SAÚDE! utiliza um sistema de design robusto e consistente baseado em Tailwind CSS, configurado para oferecer uma experiência visual coerente entre as versões web e mobile. Este guia explica como trabalhar com o design system existente, aproveitando os componentes e tokens de design predefinidos.

## O Sistema de Cores

O SAÚDE! implementa um sistema de cores sofisticado baseado em variáveis CSS que se adaptam automaticamente entre temas claro e escuro. As cores são definidas como variáveis no arquivo `globals.css` e mapeadas para classes utilitárias do Tailwind no `tailwind.config.ts`.

Para utilizar as cores do sistema em seus componentes:

TypeScript

```
// Exemplo de uso de cores do sistema
<div className="bg-primary text-typography p-4">
  <h2 className="text-accent1 text-lg
font-semibold">Acompanhamento</h2>
  <p className="text-gray2">Seus registros de humor e
atividades</p>
  <button className="bg-homebg text-offwhite px-4 py-2
rounded-lg">
    Ver detalhes
  </button>
</div>
```

O design system oferece variantes para cada cor, como `-foreground`, `-background`, e `-border`, permitindo combinações harmônicas:

TypeScript

```
// Uso de variantes de cor
<div className="bg-card border-card-border rounded-lg">
```

```
<div className="bg-accent1-background p-3">
  <span className="text-accent1">Importante!</span>
</div>
</div>
```

## Expandindo o design system

Para adicionar novos elementos ao design system mantendo a consistência:

1. Defina novas variáveis CSS em `globals.css` para ambos os temas:

```
CSS
:root {
  /* Tema claro */
  --nova-cor: #3b82f6;
}

.theme-dark {
  /* Tema escuro */
  --nova-cor: #60a5fa;
}
```

2. Adicione as cores ao `tailwind.config.ts`:

```
CSS
// Em tailwind.config.ts, na seção colors
colors: {
  // Cores existentes...
  'nova-cor': {
    DEFAULT: 'var(--nova-cor)',
    light: 'var(--nova-cor-light)',
  },
},
```

```
}
```

3. Use a nova cor em seus componentes:

HTML

```
<div className="bg-nova-cor text-white">Novo componente</div>
```

## Práticas Recomendadas

1. Consistência: Use sempre as cores e tipografia do design system em vez de valores hardcoded.
2. Componentes: Crie componentes reutilizáveis para padrões de interface comuns:

TypeScript

```
// Exemplo de componente reutilizável
export function Card({ title, children, className = "" }) {
  return (
    <div className={`bg-card border-card-border rounded-lg p-4
${className}`}>
      {title} && <h3 className="text-topicos2 mb-2">{title}</h3>
      {children}
    </div>
  );
}
```

// Uso

```
<Card title="Meus Registros">
  <p className="text-gray2">Conteúdo do card</p>
```

```
3. </Card>
```

4. Acessibilidade: Mantenha contrastes adequados entre texto e fundo:

HTML

```
// Bom contraste
<div className="bg-primary">
  <h2 className="text-primary-foreground">Título legível</h2>
</div>

// Evite
<div className="bg-primary">
  <h2 className="text-gray2">Contraste possivelmente
insuficiente</h2>

5. </div>
```

6. Animações: Use as animações predefinidas para feedback visual consistente:

HTML

```
<button className="bg-button-primary text-white
hover:animate-button-shimmer">
  Salvar

</button>
```

7. Organização: Agrupe classes Tailwind por categoria para melhor legibilidade:

TypeScript

```
// Bem organizado
<button className="
  // Layout
  px-4 py-2 rounded-lg
  // Cores
  bg-button-primary text-white
  // Efeitos
  shadow-button-soft hover:shadow-button-hover
  // Transições
  transition-all duration-300
">
  Botão Organizado
```

8. `</button>`

## Criando um Novo Componente

Esta seção fornece um guia passo a passo para criar novos componentes que se integram perfeitamente ao design system do projeto SAÚDE!. Vamos desenvolver juntos um componente de "Cartão de Status de Humor" que poderia ser usado no diário do paciente.

### 1. Estrutura do Componente

Vamos começar definindo a estrutura do nosso componente. Criaremos um arquivo chamado `mood-status-card.tsx` dentro da pasta `src/components/ui/`.

```
TypeScript
// src/components/ui/mood-status-card.tsx

import React from 'react';
import { cva } from 'class-variance-authority';

// Definimos os tipos de humor possíveis
type MoodType = 'happy' | 'neutral' | 'sad' | 'anxious' | 'calm';

// Definimos o tipo das props que nosso componente vai receber
interface MoodStatusCardProps {
  mood: MoodType;
  timestamp: Date;
  note?: string;
  onEdit?: () => void;
  className?: string;
}

// Criamos variantes de estilo baseadas no humor usando o CVA
const moodCardVariants = cva(
  // Estilos base que se aplicam a todas as variantes
```

```

    "relative p-4 rounded-lg shadow-button-soft transition-all duration-300
border",
    {
      variants: {
        mood: {
          happy: "bg-gradient-button-save border-success text-success-text",
          neutral: "bg-card border-gray2-border text-typography",
          sad: "bg-muted border-gray2-border text-typography",
          anxious: "bg-destructive/10 border-destructive/30 text-destructive",
          calm: "bg-accent1-background border-accent1/30 text-accent1",
        }
      },
      defaultVariants: {
        mood: "neutral"
      }
    }
  );

```

```

// Mapeamento de emojis para cada humor
const moodEmojis: Record<MoodType, string> = {
  happy: "😊",
  neutral: "😐",
  sad: "😞",
  anxious: "😰",
  calm: "😌"
};

```

```

// Mapeamento de descrições para cada humor
const moodDescriptions: Record<MoodType, string> = {
  happy: "Feliz",
  neutral: "Neutro",
  sad: "Triste",
  anxious: "Ansioso",
  calm: "Calmo"
};

```

```

/**
 * Componente MoodStatusCard
 *
 * Este componente exibe o estado de humor registrado pelo usuário,
 * incluindo emoji, descrição, hora e notas opcionais.
 */
export const MoodStatusCard: React.FC<MoodStatusCardProps> = ({
  mood,

```

```

timestamp,
note,
onEdit,
className = ""
}) => {
  // Formatação da data para exibir apenas hora e minutos
  const formattedTime = new Intl.DateTimeFormat('pt-BR', {
    hour: '2-digit',
    minute: '2-digit'
  }).format(timestamp);

  // Formatação da data completa para o tooltip
  const formattedDate = new Intl.DateTimeFormat('pt-BR', {
    day: '2-digit',
    month: '2-digit',
    year: 'numeric',
    hour: '2-digit',
    minute: '2-digit'
  }).format(timestamp);

  return (
    <div className={` ${moodCardVariants({ mood })} ${className} hover-lift`} >
      { /* Cabeçalho com emoji e horário */ }
      <div className="flex justify-between items-center mb-2">
        <div className="flex items-center gap-2">
          <span className="text-2xl" role="img"
aria-label={moodDescriptions[mood]}>
            {moodEmojis[mood]}
          </span>
          <span className="font-work-sans font-semibold text-topicos2">
            {moodDescriptions[mood]}
          </span>
        </div>
        <span className="text-xs text-gray2-foreground" title={formattedDate}>
          {formattedTime}
        </span>
      </div>

      { /* Nota (se existir) */ }
      {note && (
        <div className="mt-2 text-sm font-work-sans">
          <p className="line-clamp-3">{note}</p>
        </div>
      )}
    </div>
  )
}

```

```

    { /* Botão de editar (se a função de callback for fornecida) */ }
    {onEdit && (
      <button
        onClick={onEdit}
        className="absolute bottom-2 right-2 text-xs text-gray2
        hover:text-accent1 transition-colors"
        aria-label="Editar registro de humor"
      >
        <span className="mgc_edit_line text-base"></span>
      </button>
    )}
  </div>
);
};

export default MoodStatusCard;

```

## 2. Usando Tokens do Design System

Observe como o componente utiliza as classes do Tailwind que correspondem aos tokens de design:

- `bg-gradient-button-save` e `text-success-text` para o estado feliz
- `bg-card` e `text-typography` para o estado neutro
- Classes como `font-work-sans` e `font-semibold` para tipografia consistente
- Animação `hover-lift` do design system para feedback interativo
- Ícones do sistema com a classe `mgc_edit_line`

## 3. Testando o Componente

Agora vamos criar um exemplo de uso em um arquivo de teste para demonstrar como o componente pode ser utilizado:

```

TypeScript
// src/components/examples/mood-card-example.tsx

```



```

import React, { useState } from 'react';
import MoodStatusCard from '../ui/mood-status-card';

export const MoodCardExample: React.FC = () => {
  const [selectedMood, setSelectedMood] = useState<'happy' | 'neutral' | 'sad'
  | 'anxious' | 'calm'>('neutral');

  // Demonstração de dados do registro de humor
  const moodData = {
    happy: {
      timestamp: new Date(),
      note: "Hoje foi um ótimo dia! Consegui completar todas as minhas
tarefas."
    },
    neutral: {
      timestamp: new Date(Date.now() - 30 * 60 * 1000), // 30 minutos atrás
      note: "Dia comum, sem nada especial para registrar."
    },
    sad: {
      timestamp: new Date(Date.now() - 60 * 60 * 1000), // 1 hora atrás
      note: "Me senti triste após receber uma notícia desagradável."
    },
    anxious: {
      timestamp: new Date(Date.now() - 120 * 60 * 1000), // 2 horas atrás
      note: "Estou preocupado com a consulta médica amanhã."
    },
    calm: {
      timestamp: new Date(Date.now() - 180 * 60 * 1000), // 3 horas atrás
      note: "Pratiquei meditação e me senti mais tranquilo."
    }
  };

  const handleEdit = () => {
    alert(`Editando o humor: ${selectedMood}`);
  };

  return (
    <div className="p-6 max-w-md mx-auto">
      <h2 className="font-work-sans text-titulo mb-6 text-typography">
        Registro de Humor
      </h2>

      {/* Seletor de tipo de humor para demonstração */}
      <div className="flex flex-wrap gap-2 mb-6">

```

```

        {(Object.keys(moodData) as Array<keyof typeof moodData>).map((mood) =>
(
    <button
        key={mood}
        onClick={() => setSelectedMood(mood)}
        className={`px-3 py-1 rounded-full text-sm ${
            selectedMood === mood
                ? 'bg-selected text-typography font-medium'
                : 'bg-gray1 text-typography-foreground'
            }`}
    >
        {mood.charAt(0).toUpperCase() + mood.slice(1)}
    </button>
    )})
</div>

{/* Exemplo do cartão de humor */}
<MoodStatusCard
    mood={selectedMood}
    timestamp={moodData[selectedMood].timestamp}
    note={moodData[selectedMood].note}
    onEdit={handleEdit}
    className="w-full"
/>

{/* Exemplos adicionais para demonstrar diferentes estados */}
<div className="mt-8 grid grid-cols-1 md:grid-cols-2 gap-4">
    <MoodStatusCard
        mood="happy"
        timestamp={new Date()}
        className="w-full"
    />

    <MoodStatusCard
        mood="sad"
        timestamp={new Date()}
        note="Exemplo sem botão de edição"
        className="w-full"
    />
</div>
</div>
);
};

```

```
export default MoodCardExample;
```

## 4. Integração com o Design System

Para garantir que nosso componente esteja totalmente integrado com o design system, podemos adicionar suporte ao tema escuro modificando nossas variantes:

```
TypeScript
// Variantes atualizadas com suporte ao tema escuro
const moodCardVariants = cva(
  "relative p-4 rounded-lg shadow-button-soft transition-all duration-300 border",
  {
    variants: {
      mood: {
        happy: "bg-gradient-button-save border-success text-success-text",
        neutral: "bg-card border-gray2-border text-typography",
        sad: "bg-muted border-gray2-border text-typography",
        anxious: "bg-destructive/10 border-destructive/30 text-destructive",
        calm: "bg-accent1-background border-accent1/30 text-accent1",
      }
    },
    defaultVariants: {
      mood: "neutral"
    }
  }
);
```

Como o sistema já usa variáveis CSS que se adaptam automaticamente entre temas claro e escuro, não precisamos fazer alterações adicionais.

## 5. Documentação do Componente

É importante documentar bem os componentes. Adicionaremos comentários JSDoc ao componente:

TypeScript

```
/**
 * MoodStatusCard
 *
 * Componente que exibe o estado de humor de um usuário em formato de cartão,
 * com suporte a diferentes estados emocionais.
 *
 * @component
 * @param {Object} props - As propriedades do componente
 * @param {MoodType} props.mood - O tipo de humor ('happy', 'neutral', 'sad',
 'anxious', 'calm')
 * @param {Date} props.timestamp - A data e hora do registro de humor
 * @param {string} [props.note] - Uma nota opcional associada ao registro de
 humor
 * @param {Function} [props.onEdit] - Callback opcional para editar o registro
 * @param {string} [props.className] - Classes adicionais para estilização
 *
 * @example
 * // Cartão básico
 * <MoodStatusCard mood="happy" timestamp={new Date()} />
 *
 * // Cartão com nota e callback de edição
 * <MoodStatusCard
 *   mood="anxious"
 *   timestamp={new Date()}
 *   note="Estou me sentindo ansioso sobre a apresentação de amanhã."
 *   onEdit={() => openEditModal(moodRecord.id)}
 * />
 */
```

## 6. Integração com a Aplicação

Finalmente, vamos integrar nosso componente a uma página existente no aplicativo, como o diário do paciente:

TypeScript

```
// src/pages/patient/diary/Diary.tsx (modificado para incluir nosso componente)

import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import MoodStatusCard from '@components/ui/mood-status-card';
import { DiaryService } from '@api/services/DiaryService';
```

```

// Suponha que tenhamos um tipo para entradas de diário
interface DiaryEntry {
  id: string;
  date: string;
  mood: 'happy' | 'neutral' | 'sad' | 'anxious' | 'calm';
  note?: string;
}

export const Diary: React.FC = () => {
  const { id } = useParams<{ id: string }>();
  const [entries, setEntries] = useState<DiaryEntry[]>([]);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    const fetchDiaryEntries = async () => {
      try {
        setIsLoading(true);
        // Suponha que tenhamos um serviço para buscar entradas de diário
        const response = await DiaryService.getDiaryEntries(id || '');
        setEntries(response.data);
      } catch (error) {
        console.error('Erro ao carregar entradas do diário:', error);
      } finally {
        setIsLoading(false);
      }
    };

    fetchDiaryEntries();
  }, [id]);

  const handleEditEntry = (entryId: string) => {
    // Lógica para abrir modal de edição ou navegar para página de edição
    console.log(`Editando entrada: ${entryId}`);
  };

  return (
    <div className="p-4">
      <h1 className="font-work-sans text-titulo mb-6">Meu Diário</h1>

      {isLoading ? (
        <div className="flex justify-center">
          <div className="animate-spin rounded-full h-8 w-8 border-3
border-primary/30 border-t-primary"></div>

```

```

    </div>
  ) : entries.length === 0 ? (
    <div className="text-center p-6 bg-card rounded-lg shadow-button-soft">
      <p className="text-gray2">Você ainda não tem registros no diário.</p>
      <button className="mt-4 bg-gradient-button-primary text-white px-4
py-2 rounded-md shadow-button-soft hover:shadow-button-hover transition-all">
        Criar Primeiro Registro
      </button>
    </div>
  ) : (
    <div className="space-y-4">
      {entries.map(entry => (
        <MoodStatusCard
          key={entry.id}
          mood={entry.mood}
          timestamp={new Date(entry.date)}
          note={entry.note}
          onEdit={() => handleEditEntry(entry.id)}
          className="w-full"
        />
      ))}
    </div>
  )}
</div>
);
};

export default Diary;

```

## 7. Considerações sobre Responsividade

Nosso componente é naturalmente responsivo graças às classes do Tailwind, mas podemos adicionar alguns ajustes específicos:

```

TypeScript
// Dentro do componente MoodStatusCard, modificamos o JSX para ser mais
responsivo

return (
  <div className={` ${moodCardVariants({ mood })} ${className} hover-lift`} >

```

```

    <div className="flex flex-col sm:flex-row sm:justify-between
sm:items-center mb-2">
      <div className="flex items-center gap-2 mb-2 sm:mb-0">
        <span className="text-2xl sm:text-3xl" role="img"
aria-label={moodDescriptions[mood]}>
          {moodEmojis[mood]}
        </span>
        <span className="font-work-sans font-semibold text-topicos2">
          {moodDescriptions[mood]}
        </span>
      </div>
      <span className="text-xs text-gray2-foreground" title={formattedDate}>
        {formattedTime}
      </span>
    </div>

    { /* Resto do componente permanece igual */ }
  </div>
);

```

## 8. Performance e Acessibilidade

Para garantir que nosso componente seja performático e acessível:

1. Usamos `role="img"` e `aria-label` para o emoji
2. Fornecemos texto descritivo para o botão de edição
3. Adicionamos `title` para mostrar a data completa no hover
4. Limitamos a exibição da nota com `line-clamp-3` para evitar cartões muito grandes

## Configurações Específicas do Google OAuth

Para o funcionamento adequado do login via Google OAuth, são necessários dois Google Apps configurados sob o mesmo projeto (SAUDE):

### Google Android App

Este app é do tipo `android` e tem como única finalidade o cadastro do SHA-1 do aplicativo. Isso é crucial para o login mobile, que requer a geração de um token

em vez de um código. O `clientId` e `secret` deste app não são utilizados; ele serve exclusivamente para o registro do SHA-1 no projeto.

## Google Web App

Este app deve ser do tipo `Web App` e será utilizado tanto para o login na versão web quanto na versão mobile. É necessário cadastrar as URLs de redirecionamento para o caso de login web, onde o servidor chama o cliente na web (atualmente, somente `localhost`). O `clientId` e `secret` deste app devem ser utilizados como variáveis de ambiente tanto neste projeto (frontend) quanto no servidor (backend). O `clientSecret` é necessário apenas no servidor para o login web.

## Capacitor (Builds para Mobile)

O Capacitor é utilizado para empacotar o projeto React em um aplicativo nativo para Android.

### Gerar Nova Build

Para gerar uma nova build do projeto, execute:

```
npm run build
```

### Copiar para o Android

Para copiar a build gerada para o ambiente Android, utilize:

```
npx cap sync android
```

### Abrir no Android Studio

Para abrir o projeto no Android Studio e continuar com o desenvolvimento ou testes no ambiente nativo:

```
npx cap open android
```



## Testar no Android Studio

Dentro do Android Studio, siga os passos para testar o aplicativo:

1. Clique no botão verde de "Run" (geralmente um triângulo).
2. Escolha um dispositivo:
  - **Dispositivo real:** Conecte seu celular via USB e ative o modo de desenvolvedor.
  - **Emulador:** Selecione um emulador configurado.
3. Aguarde a instalação e abertura do aplicativo no dispositivo ou emulador.

## (Opcional) Gerar o Bundle para a Play Store

Para gerar o Android App Bundle (.aab) para publicação na Google Play Store:

1. No Android Studio, vá em **Build > Generate Signed Bundle / APK**.
2. Selecione **Android App Bundle (.aab)**.