Documentação do Aplicativo de Segurança para Mulheres

Visão Geral do Projeto

O aplicativo tem como objetivo aumentar a segurança de mulheres que precisam se deslocar à noite em áreas com pouca iluminação no Brasil. O conceito central é um "guardião digital" que monitora sons do ambiente e identifica possíveis situações de perigo para alertar contatos de confiança.

Funcionalidades Principais

1. Monitoramento de áudio contínuo em segundo plano
2. Detecção de sons que indiquem perigo (gritos, sons de confronto, pedidos de ajuda)
3. Envio automático de alertas com localização GPS para contatos pré-selecionados
4. Opção de ativação manual ou automática baseada em condições (horário, localização)

Considerações Importantes

- **Privacidade:** O app precisará ser projetado para não gravar conversas normais, apenas reagir a padrões sonoros específicos que indiquem perigo
- **Consumo de bateria:** Monitoramento contínuo pode consumir muita bateria, será necessário otimizar
- **Falsos positivos:** O sistema precisa ser calibrado para minimizar falsos alarmes
- **Conectividade:** Deve funcionar mesmo com conexão limitada de internet

Tecnologias Sugeridas

Para desenvolvimento do aplicativo

- **Framework multiplataforma:** Flutter (com Dart como linguagem principal)
- **Backend:** Firebase ou AWS para gerenciar dados, autenticação e envio de notificações

Para o reconhecimento de áudio

- **Machine Learning:** TensorFlow Lite ou CoreML para processamento de áudio no dispositivo
- **Modelos pré-treinados:** Adaptar modelos existentes de reconhecimento de sons de perigo/emergência
- **Edge computing:** Processamento local para minimizar latência e dependência de internet

Para a localização e alertas

- **APIs de geolocalização:** Google Maps API ou equivalentes para localização precisa
- **SMS fallback:** Sistema para enviar SMS quando não há conexão com a internet

- **Webhooks:** Para integração com serviços de emergência ou plataformas como WhatsApp

Para otimização de bateria

- **Algoritmos adaptativos:** Ajuste da sensibilidade e frequência de monitoramento baseado no nível de bateria
- **Ativação contextual:** Usar sensores do telefone (luz ambiente, movimento) para ativar o monitoramento apenas quando necessário

Para a interface do usuário

- **Design simplificado:** Interface intuitiva com botão de pânico facilmente acessível
- **Feedback tátil:** Confirmação discreta quando o app está ativo (vibração sutil)
- **Modo escuro:** Para não chamar atenção em ambientes com pouca luz

Para segurança dos dados

- **Criptografia:** Para proteger informações sensíveis como localização e contatos
- **Armazenamento local:** Manter dados pessoais principalmente no dispositivo

MVP (Produto Mínimo Viável)

Funcionalidades essenciais

- Monitoramento básico de áudio para detectar gritos ou palavras-chave de perigo
- Envio de SMS com localização para até 3 contatos de emergência pré-cadastrados
- Botão de pânico manual para ativação imediata do alerta
- Interface simples com modo claro/escuro

Escopo técnico reduzido

- Foco em uma única plataforma inicialmente (Android, por ter maior penetração no Brasil)
- Detecção de áudio simplificada (poucos padrões sonoros básicos)
- Geolocalização básica sem mapeamento detalhado
- Armazenamento local dos contatos de emergência

Fluxo de uso simplificado

- Cadastro rápido (nome, número de celular)
- Adição de contatos de emergência (nome e número)
- Botão grande para ativar/desativar o monitoramento
- Configuração opcional de horários para ativação automática

Limitações aceitáveis no MVP

- Maior consumo de bateria (otimização viria em versões posteriores)
- Possibilidade de alguns falsos positivos
- Funcionamento apenas com conexão de dados ou SMS
- Interface mais básica

Detecção de Áudio Simplificada no MVP

Reconhecimento de padrões sonoros básicos

- Detecção de volume alto (gritos)
- Identificação de palavras-chave de emergência em português ("socorro", "ajuda", "polícia")
- Detecção de sons abruptos e intensos (indicativos de confronto)

Implementação técnica simplificada

- Uso de algoritmos de processamento de áudio mais leves
- Análise de frequência e amplitude do som para identificar padrões de gritos
- Pequeno modelo de machine learning pré-treinado com dataset limitado de sons de emergência

Funcionamento prático

- O app captaria áudio em intervalos curtos (ex: amostragens de 2-3 segundos)
- Análise seria feita no próprio dispositivo, sem enviar dados para a nuvem
- Ao identificar um padrão suspeito, o app faria uma verificação adicional por 5-10 segundos
- Se o padrão persistir ou se intensificar, o alerta seria disparado

Calibração básica

- Opção para ajustar a sensibilidade (baixa, média, alta)
- Definição de um limiar de intensidade sonora para evitar os falsos positivos mais óbvios
- Possibilidade de o usuário "treinar" o app com exemplos de sua própria voz pedindo socorro

Estrutura do Projeto

```
seguranca-app/
├── android/                    # Configurações específicas do Android
├── assets/
│   ├── images/                 # Ícones e imagens
│   └── sounds/                 # Sons de alerta e amostras para treinamento
├── lib/
│   ├── main.dart               # Ponto de entrada do aplicativo
│   ├── config/
│   │   ├── constants.dart      # Constantes e configurações globais
│   │   └── theme.dart          # Tema e estilos da UI
│   ├── modules/
│   │   ├── audio_detection/
│   │   │   ├── services/
│   │   │   │   ├── audio_capture_manager.dart
│   │   │   │   ├── audio_processor.dart
│   │   │   │   └── sound_pattern_detector.dart
│   │   │   ├── models/
│   │   │   │   └── sound_pattern.dart
│   │   │   └── utils/
│   │   │       └── audio_permissions.dart
│   │   ├── location/
│   │   │   ├── services/
│   │   │   │   ├── location_tracker.dart
│   │   │   │   └── geocoding_util.dart
│   │   │   └── models/
│   │   │       └── location_data.dart
│   │   ├── alerts/
│   │   │   ├── services/
│   │   │   │   ├── alert_manager.dart
│   │   │   │   └── sms_service.dart
│   │   │   └── models/
│   │   │       ├── alert.dart
│   │   │       └── emergency_contact.dart
│   │   └── user/
│   │       ├── services/
│   │       │   ├── user_service.dart
│   │       │   └── storage_service.dart
│   │       └── models/
│   │           ├── user.dart
│   │           └── user_settings.dart
│   ├── screens/
│   │   ├── onboarding/
│   │   │   ├── intro_screen.dart
│   │   │   └── permissions_screen.dart
│   │   ├── auth/
│   │   │   ├── register_screen.dart
```

```
│   │   │   └── login_screen.dart
│   │   ├── home/
│   │   │   ├── home_screen.dart
│   │   │   └── monitoring_status.dart
│   │   ├── contacts/
│   │   │   ├── contacts_list_screen.dart
│   │   │   └── add_contact_screen.dart
│   │   └── settings/
│   │       ├── settings_screen.dart
│   │       ├── audio_sensitivity_screen.dart
│   │       └── schedule_screen.dart
│   ├── utils/
│   │   ├── permissions_manager.dart   # Gerenciamento central de permissões
│   │   ├── validators.dart           # Validação de formulários
│   │   └── date_time_utils.dart      # Utilitários para manipulação de datas/horas
│   └── widgets/
│       ├── emergency_button.dart     # Botão de pânico personalizado
│       ├── status_indicator.dart     # Indicador visual de status do monitoramento
│       ├── contact_card.dart         # Card para exibir contato de emergência
│       └── sensitivity_slider.dart    # Slider para ajustes de sensibilidade
├── test/
│   ├── unit/
│   │   ├── audio_detection_test.dart
│   │   ├── location_test.dart
│   │   └── alert_test.dart
│   └── widget/
│       ├── home_screen_test.dart
│       └── emergency_button_test.dart
└── pubspec.yaml                      # Dependências
```

Fluxos de Funcionamento

Fluxo de Inicialização

1. Usuário abre o app pela primeira vez
2. Visualiza telas de onboarding
3. Solicita permissões necessárias (microfone, localização, SMS)
4. Cria conta básica ou pula etapa
5. Configura contatos de emergência

Fluxo de Monitoramento

1. Usuário ativa monitoramento na tela inicial
2. `AudioCaptureManager` inicia captura em intervalos

3. `AudioProcessor` analisa amostras de áudio
4. `SoundPatternDetector` verifica padrões de perigo
5. Interface atualiza status de monitoramento

Fluxo de Alerta

1. Padrão de som de perigo é detectado (ou botão de pânico é pressionado)
2. `AlertManager` é acionado
3. `LocationTracker` obtém localização atual
4. `GeocodingUtil` converte coordenadas em endereço legível
5. `SMSService` envia mensagens para contatos de emergência
6. Interface exibe confirmação de alerta enviado

Fluxo de Configuração

1. Usuário acessa tela de configurações
2. Ajusta sensibilidade de detecção
3. Define horários para ativação automática
4. Configura mensagem personalizada para alertas
5. `UserService` salva configurações no armazenamento local

Detalhes do AudioCaptureManager

O `AudioCaptureManager` é responsável por controlar todo o ciclo de captura de áudio do aplicativo:

Inicialização e configuração

- Quando o usuário ativa o monitoramento, o `AudioCaptureManager` é inicializado
- Ele configura parâmetros como taxa de amostragem (ex: 16kHz), formato de áudio (ex: PCM 16-bit) e tamanho do buffer
- Verifica e solicita permissões de acesso ao microfone se necessário

Captura em intervalos

- Em vez de fazer uma gravação contínua (que consumiria muita bateria), o manager implementa uma estratégia de captura em intervalos
- Por exemplo: captura 2-3 segundos de áudio, processa, pausa por 1-2 segundos, e repete
- Esse ciclo ajuda a economizar recursos do dispositivo

Adaptação inteligente

- Conforme o nível de bateria diminui, o manager aumenta automaticamente o intervalo entre capturas
- Em ambientes mais ruidosos, pode reduzir o intervalo para capturar com mais frequência
- Pode usar os sensores do telefone (como o acelerômetro) para detectar se o usuário está em movimento ou parado, ajustando a frequência de captura

Modo de economia de energia

- Durante períodos de inatividade detectados (quando o telefone está imóvel por muito tempo), aumenta significativamente os intervalos
- Retorna ao monitoramento normal quando detecta movimento

Comunicação com outros componentes

- Após cada captura, envia os dados para o `AudioProcessor`
- Recebe feedback do `SoundPatternDetector` para ajustar seus parâmetros
- Implementa um sistema de eventos para notificar a interface sobre o status do monitoramento

Gerenciamento de recursos

- Libera recursos do microfone quando o monitoramento é pausado ou desativado
- Implementa lógica para evitar conflitos com outros apps que possam estar usando o microfone

Soluções para Não Perder Sinais de Emergência Entre Intervalos

Captura com sobreposição

- Em vez de pausas completas, implementar uma sobreposição de capturas
- Exemplo: capturar 3 segundos, processar, mas iniciar nova captura após 1,5 segundos (50% de sobreposição)
- Isso garante que nenhum momento fique sem monitoramento

Buffer circular contínuo

- Manter um buffer circular de áudio em execução constante
- O áudio é gravado continuamente em um buffer de curta duração (5-10 segundos)
- O processamento ocorre em intervalos sobre esse buffer em movimento
- Se um alerta for detectado, o buffer completo é preservado como evidência

Detecção de volume em segundo plano

- Mesmo durante "intervalos", manter um monitor simples de amplitude de som
- Se um som alto repentino for detectado, iniciar imediatamente uma captura completa

- Isso consome menos recursos que o processamento completo, mas não deixa brechas

Modo de dois níveis

- Processamento leve contínuo para detecção básica (palavras-chave, volume alto)
- Processamento pesado em intervalos para análise detalhada
- Se o processamento leve detectar algo suspeito, aciona imediatamente o processamento completo

Análise de contexto sonoro

- Implementar detecção de mudanças bruscas no ambiente sonoro
- Se o padrão de som mudar drasticamente (silêncio repentino após sons altos), isso também pode ser um gatilho de alerta

Configuração do VS Code para o Projeto

Extensões essenciais

- Flutter (oficial da Google/Dart team)
- Dart (suporte à linguagem)
- Flutter Widget Snippets (atalhos para widgets comuns)
- Awesome Flutter Snippets (snippets adicionais)
- Material Icon Theme (para melhorar identificação visual de arquivos)
- Error Lens (visualização de erros diretamente no código)

Configurações recomendadas

- Habilitar formatação automática ao salvar (editor.formatOnSave: true)
- Configurar o Dart formatter (dart.lineLength: 80)
- Ativar análise de código em tempo real

Workflow de desenvolvimento

- Usar o Command Palette (Ctrl+Shift+P) para comandos Flutter comuns
- Configurar o Flutter Outline (painel lateral) para visualização da estrutura de widgets
- Utilizar o terminal integrado para comandos flutter e testes

Iniciando o projeto

- Criar o projeto via terminal integrado: `flutter create --org com.seunome seguranca_app`
- Usar o VS Code para gerar a estrutura de pastas definida
- Configurar o pubspec.yaml com as dependências iniciais

Depuração com VS Code

- Iniciar depuração com F5 (selecionar dispositivo quando solicitado)
- Usar o painel de DevTools acessível diretamente do VS Code
- Configurar Hot Reload automático (r) e Hot Restart (R) durante depuração

Recursos específicos para nosso projeto

- Usar o Flutter Performance para monitorar o consumo de recursos do app
- Configurar a visualização de logs para depurar o processamento de áudio
- Usar o perfil de benchmark para testar o desempenho em diferentes dispositivos