vamos fazer então o Sprint 1 (Mobile + NLP básico): telas de gravação/edição, speech_to_text integrado, envio para /processar-fala e renderização dos campos extraídos/alertas; seguir guias oficiais de Flutter para navegação/estado e do plugin para permissões e callbacks.

A implementação do Sprint 1 deve usar Flutter no mobile com o plugin speech_to_text para reconhecimento de voz local, estado simples com Provider e chamadas HTTP via pacote http para o endpoint /processar-fala no FastAPI, cobrindo telas de gravação/edição, permissões e renderização dos campos extraídos e alertas retornados pela API.[1] [2] [3] [4] [5]

Stack do Sprint 1

- Mobile: Flutter com UI declarativa e hot reload para iterar rápido nas telas de gravação e revisão dos dados extraídos.[3] [6]
- Voz on-device: plugin speech_to_text para inicializar, escutar, tratar eventos de resultado/erro e encerrar a sessão de reconhecimento.[4] [7]
- Networking: pacote http do Dart para POST JSON ao backend FastAPI e consumo das respostas tipadas.[2] [1]
- Estado: Provider/ChangeNotifier para compartilhar o texto reconhecido e o resultado do NLP entre as telas.[8] [9]
- Permissões: permission_handler para pedir microfone (e abrir configurações se negado) e INTERNET no AndroidManifest.[10] [11] [2]

Passo 1: Criar projeto Flutter

- Comando: flutter create agrovoz_mobile, abrir no IDE e validar execução de app padrão em dispositivo/emulador.[6] [3]
- Estrutura mínima: lib/main.dart, lib/screens/record_screen.dart, lib/screens/review_screen.dart, lib/services/speech_service.dart, lib/services/api_service.dart, lib/state/session_provider.dart.[9] [6]

Passo 2: Adicionar dependências

- Adicionar pacotes: speech_to_text, http, provider e permission_handler com flutter pub add. [11] [1] [9] [4]
- Referência: http fornece funções Future-based e requer INTERNET no AndroidManifest; provider é a abordagem recomendada para estado simples; speech_to_text expõe APIs cross-platform. [2] [9] [4]

Exemplo (pubspec.yaml – trechos relevantes):

- dependencies: provider, speech_to_text, http, permission_handler. [1] [11] [9] [4]

Passo 3: Permissões e configurações

- Android: adicionar no AndroidManifest para networking e as permissões de microfone necessárias pelo speech_to_text. [4] [2]
- iOS: adicionar chaves de Info.plist para NSMicrophoneUsageDescription e NSSpeechRecognitionUsageDescription conforme instruções do plugin. [4]
- Em runtime: usar permission_handler para solicitar Permission.microphone e lidar com denial/permanently denied (abrir configurações se preciso). [10] [11]

Passo 4: Estado com Provider

- Criar um ChangeNotifier (SessionProvider) com campos recognizedText, isListening, extractedData, alerts e métodos para iniciar/parar escuta e para enviar texto ao backend. [9]
- Registrar ChangeNotifierProvider no topo do app (MultiProvider no main.dart) para consumir estado nas telas. [9]

Passo 5: Integração speech_to_text

- Inicializar SpeechToText, verificar disponibilidade no dispositivo e iniciar escuta com locale pt_BR, tratando onResult para acumular recognizedWords. [4]
- Controlar ciclo de vida: iniciar/pausar/parar, checar hasPermission e escutar eventos de status/erro, atualizando o Provider. [4]

Exemplo (trechos – serviço de voz):

- Inicialização: await speech.initialize() e verificação de permissão. [4]
- Escuta: await speech.listen(onResult: callback, localeId: 'pt_BR') e stop/cancel ao finalizar. [4]

Passo 6: Chamada HTTP para /processar-fala

- Usar http.post com Content-Type application/json, serializando { texto, usuario_id } e parseando o body em um modelo de resposta (dados_extraidos, validacao, confianca, sugestoes). [1] [2]

- Em Android, lembrar da permissão INTERNET já adicionada; em macOS, habilitar entitlement de network se for alvo desktop.[2]

Exemplo (ApiService – trechos essenciais):

- Import 'package:http/http.dart' as http; e método Future<RespostaNLP> processarFala(String texto, String usuarioId).[1] [2]
- Tratar erros: statusCode != 200, timeouts e jsonDecode, retornando objeto tipado para a tela Review.[2]

Passo 7: Telas do Sprint 1

- RecordScreen: botão grande de microfone, indicador de escuta, texto reconhecido em tempo real e botão "Enviar" para chamar a API; usar Consumer<SessionProvider> para reatividade.[6] [9]
- ReviewScreen: exibir campos extraídos (pessoa, atividade, cultura, talhão, valor), lista de alertas/erros e botões "Confirmar"/"Editar novamente"; navegação via Navigator push/pop.[6] [9]

Passo 8: Fluxo UX/Controle

- Fluxo: pedir permissão microfone → iniciar escuta → renderizar texto parcial → parar → permitir edição → enviar para API → mostrar resultado NLP/alertas → confirmar.[11] [4]
- Estados: loading ao chamar API, mensagens de erro/sem rede e retry; logs para depuração enquanto itera.[9] [2]

Passo 9: Navegação e gerenciamento de estado simples

- Usar navegação padrão (Navigator) e Provider para compartilhar o texto e a resposta do NLP; a própria documentação recomenda Provider para apps simples.[6] [9]
- Alternativamente, manter estado local com StatefulWidget para a RecordScreen e mover o resultado para Provider só ao confirmar envio.[8] [9]

Passo 10: Testes manuais no emulador/dispositivo

- Validar o reconhecimento de voz em pt_BR, confirmar que recognizedWords atualiza em tempo real e que a API retorna JSON com dados e alertas.[2] [4]
- Checar permissões negadas: abrir configurações com openAppSettings se permanentlyDenied e exibir instruções ao usuário.[10] [11]

Snippets essenciais

Main + Provider (trecho):

- runApp com MultiProvider e ChangeNotifierProvider(create: (_) ⇒ SessionProvider()).[9]

Speech (trecho):

- final speech = SpeechToText(); await speech.initialize(); await speech.listen(onResult: ..., localeId: 'pt_BR'); speech.stop();[4]

HTTP (trecho):

- final res = await http.post(Uri.parse('$baseUrl/api/v1/processar-fala'), headers: {'Content-Type':'application/json'}, body: jsonEncode({...})); parse com jsonDecode.[1] [2]

Permissões (trecho):

- final status = await Permission.microphone.request(); if (status.isPermanentlyDenied) openAppSettings();[11] [10]

AndroidManifest:

- e permissões de áudio exigidas pelo speech_to_text.[2] [4]

iOS Info.plist:

- NSMicrophoneUsageDescription e NSSpeechRecognitionUsageDescription com mensagens claras ao usuário.[4]

Guia de referência rápida

- Flutter docs gerais e API: criação do app, widgets, navegação e build.[12] [6]
- Cookbook de networking com http: adicionar dependência, permissões, requisições e parsing.[2]
- speech_to_text plugin: instalação, inicialização, listen/stop e requisitos de plataforma.[7] [4]
- Estado recomendado (Provider) e guia de app state simples.[8] [9]
- FastAPI para endpoint /processar-fala no backend já em execução local, testável via Postman.[5]

Seguindo esses passos, o Sprint 1 entrega um app funcional com gravação/edição, transcrição local confiável, envio ao backend e exibição de campos extraídos/alertas, com arquitetura simples e documentação oficial sustentando cada decisão técnica.[5] [1] [9] [2] [4]

⁂

1. https://pub.dev/packages/http
2. https://docs.flutter.dev/cookbook/networking/fetch-data
3. https://flutter.dev
4. https://pub.dev/packages/speech_to_text
5. https://realpython.com/fastapi-python-web-apis/
6. https://docs.flutter.dev
7. https://pub.dev/packages/speech_to_text/versions
8. https://docs.flutter.dev/get-started/fundamentals/state-management
9. https://docs.flutter.dev/data-and-backend/state-mgmt/simple
10. https://www.scaler.com/topics/permission-handler-flutter/

11. https://pub.dev/packages/permission_handler
12. https://api.flutter.dev
13. https://fluttergems.dev/http-client-utilities/
14. https://www.alura.com.br/artigos/obter-dados-internet-flutter-usando-http
15. https://blog.joflee.com/beginners-guide-to-flutter-networking-with-http-package
16. https://www.geeksforgeeks.org/flutter/flutter-state-management-provider/
17. https://github.com/dart-lang/http