

# Complete Development Prompts for Kiro - Fish Ledger App

## Project Overview for Kiro

**\*\*App Name\*\*:** Fish Ledger  
**\*\*Purpose\*\*:** Voice-activated digital bookkeeping for fish sellers in Ghana  
**\*\*Technology Stack\*\*:** Flutter + Firebase + Google Cloud Speech-to-Text  
**\*\*Timeline\*\*:** 27 days to working MVP  
**\*\*Target\*\*:** Proof-of-concept with 60%+ transaction accuracy

## PROJECT INITIALIZATION PROMPTS

### Prompt 1: Project Setup and Structure

Create a new Flutter project called "fish\_ledger" with the following requirements:

PROJECT STRUCTURE:

```
fish_ledger/
├── lib/
│   ├── main.dart
│   ├── models/
│   │   ├── transaction.dart
│   │   ├── user_profile.dart
│   │   └── audio_segment.dart
│   ├── services/
│   │   ├── audio_service.dart
│   │   ├── speech_recognition_service.dart
│   │   ├── transaction_detection_service.dart
│   │   └── firebase_service.dart
│   ├── screens/
│   │   ├── home_screen.dart
│   │   ├── transaction_list_screen.dart
│   │   ├── settings_screen.dart
│   │   └── onboarding_screen.dart
│   ├── widgets/
│   │   ├── transaction_card.dart
│   │   ├── daily_summary.dart
│   │   └── listening_indicator.dart
│   └── utils/
│       ├── constants.dart
│       ├── pattern_matcher.dart
│       └── helpers.dart
```

DEPENDENCIES TO ADD (pubspec.yaml):

```
dependencies:
  flutter:
    sdk: flutter
  # Firebase
  firebase_core: ^2.24.2
  cloud_firestore: ^4.13.6
  firebase_auth: ^4.15.3
  firebase_analytics: ^10.7.4
```

```
# Audio
record: ^5.0.4
permission_handler: ^11.1.0
path_provider: ^2.1.1
```

```
# HTTP & API
http: ^1.1.2
```

```
# State Management
provider: ^6.1.1
```

```
# UI
intl: ^0.18.1
```

```
# Storage
shared_preferences: ^2.2.2
```

#### FIREBASE CONFIGURATION:

1. Set up Firebase project in console
2. Add Android configuration (google-services.json)
3. Add iOS configuration (GoogleService-Info.plist)
4. Enable Firestore Database
5. Enable Authentication (Anonymous for MVP)
6. Set up Firestore security rules for development:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId}/{document=**} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

#### INITIAL SETUP REQUIREMENTS:

- Minimum SDK: Android 21, iOS 12
- Permissions: RECORD\_AUDIO, INTERNET, WRITE\_EXTERNAL\_STORAGE
- Support for background audio processing
- Enable offline persistence for Firestore

Generate the complete project structure with all necessary configuration files.  
...

---

#### ## CORE COMPONENT PROMPTS

##### ### Prompt 2: Data Models

...

Create the following data models for the Fish Ledger app:

1. TRANSACTION MODEL (lib/models/transaction.dart):

```
class Transaction {
  String id;
```

```

DateTime timestamp;
String fishType; // "tilapia", "tuna", "mackerel", etc.
double amount; // Price in Ghana Cedis
int quantity; // Number of fish
double confidence; // 0.0 to 1.0
String rawTranscript; // What was said
TransactionStatus status; // auto, manual, corrected
String sellerId;

```

```

// Methods:
- toJson() for Firestore
- fromJson() for Firestore
- copyWith() for updates
}

```

```
enum TransactionStatus { auto, manual, corrected, deleted }
```

2. USER PROFILE MODEL (lib/models/user\_profile.dart):

```

class UserProfile {
  String userId;
  String name;
  String phoneNumber;
  String marketLocation;
  List<String> commonFishTypes;
  DateTime createdAt;
  Map<String, dynamic> settings;

```

```

// Methods:
- toJson()
- fromJson()
- updateSettings()
}

```

3. AUDIO SEGMENT MODEL (lib/models/audio\_segment.dart):

```

class AudioSegment {
  String id;
  DateTime timestamp;
  String filePath;
  double durationSeconds;
  double noiseLevel;
  int speakerCount;
  bool processed;
  String? transcript;

```

```

// Methods:
- toJson()
- fromJson()
- markAsProcessed()
}

```

REQUIREMENTS:

- All models must be immutable where possible
- Include proper null safety
- Add validation in fromJson methods
- Include helpful toString() methods for debugging
- Add equality operators (== and hashCode)

Generate complete model classes with all methods implemented.

...

---

### Prompt 3: Audio Recording Service

...

Create a comprehensive audio recording service (lib/services/audio\_service.dart) with the following requirements:

CLASS: AudioService

FEATURES:

1. Continuous background audio recording
2. Circular buffer (keep last 30 seconds)
3. Voice Activity Detection (VAD)
4. Battery-optimized recording
5. Audio quality monitoring

KEY METHODS:

- startListening() → Start continuous recording
- stopListening() → Stop and cleanup
- pauseListening() → Temporarily pause
- resumeListening() → Resume from pause
- getAudioStream() → Stream<Uint8List> of audio data
- getCurrentNoiseLevel() → double (0.0 to 1.0)
- isVoiceDetected() → bool

TECHNICAL REQUIREMENTS:

- Use 'record' package for audio capture
- Sample rate: 16000 Hz (optimal for speech)
- Encoding: PCM 16-bit
- Mono channel (not stereo)
- Buffer size: 8192 bytes
- Implement Voice Activity Detection using amplitude threshold
- Save audio chunks as temporary WAV files
- Auto-cleanup old audio files (>24 hours)
- Handle permission requests
- Handle interruptions (phone calls, other apps)

BATTERY OPTIMIZATION:

- Only process when amplitude > threshold
- Reduce processing when phone is stationary
- Stop recording during device sleep (optional)

ERROR HANDLING:

- Permission denied
- Microphone in use by another app
- Storage full
- Audio device errors

EXAMPLE USAGE:

```
final audioService = AudioService();  
await audioService.startListening();
```

```
audioService.getAudioStream().listen((audioData) {  
  // Process audio  
});
```

Generate the complete AudioService class with proper error handling and state management.  
...

--

### ### Prompt 4: Speech Recognition Service

...

Create a Speech Recognition Service (lib/services/speech\_recognition\_service.dart) that interfaces with Google Cloud Speech-to-Text API:

CLASS: SpeechRecognitionService

#### FEATURES:

1. Convert audio to text using Google Cloud Speech-to-Text
2. Support for Ghanaian English accent
3. Handle code-switching (English + Twi)
4. Real-time streaming recognition
5. Batch processing for saved audio files

#### CONFIGURATION:

- API Endpoint: <https://speech.googleapis.com/v1/speech:recognize>
- Language codes: en-GH (Ghana English), tw-GH (Twi if available, else en-US)
- Audio encoding: LINEAR16
- Sample rate: 16000 Hz
- Enable automatic punctuation
- Enable word-level confidence

#### KEY METHODS:

- recognizeAudio(Uint8List audioData) → Future<RecognitionResult>
- streamRecognition(Stream<Uint8List> audioStream) → Stream<String>
- setLanguagePreference(String languageCode)
- getConfidenceScore() → double

#### DATA CLASSES:

```
class RecognitionResult {  
  String transcript;  
  double confidence;  
  List<WordInfo> words;  
  String languageCode;  
  DateTime timestamp;  
}
```

```
class WordInfo {  
  String word;  
  double confidence;  
  Duration startTime;  
  Duration endTime;  
}
```

#### OPTIMIZATION:

- Cache API credentials securely

- Implement retry logic with exponential backoff
- Handle rate limiting
- Process audio in chunks (max 60 seconds per request)
- Queue requests during high load

#### ERROR HANDLING:

- Network errors
- API quota exceeded
- Invalid audio format
- Authentication failures
- Timeout errors

#### COST OPTIMIZATION:

- Track API usage
- Only process when high confidence of speech
- Use free tier efficiently (60 minutes/month)

#### API KEY MANAGEMENT:

- Store API key in environment variables or Firebase Remote Config
- Never hardcode in source code
- Include instructions for developers to add their own key

Generate the complete SpeechRecognitionService with proper error handling, retries, and optimization.

...

---

#### ### Prompt 5: Transaction Detection Service

...

Create a Transaction Detection Service (lib/services/transaction\_detection\_service.dart) that analyzes transcripts and extracts transaction information:

CLASS: TransactionDetectionService

PURPOSE: Parse speech transcripts to identify and extract fish market transactions

#### FEATURES:

1. Detect transaction patterns in conversations
2. Extract fish types, prices, quantities
3. Implement transaction state machine
4. Calculate confidence scores
5. Filter false positives

#### TRANSACTION STATE MACHINE:

```
enum TransactionState {
  idle,          // No transaction in progress
  priceInquiry,  // Customer asking price
  priceGiven,    // Seller stating price
  negotiation,   // Price bargaining
  payment,       // Payment confirmation
  completed      // Transaction finalized
}
```

#### KEY METHODS:

- processTranscript(String transcript, String speaker) → TransactionEvent?

- extractPrice(String text) → double?
- extractFishType(String text) → String?
- extractQuantity(String text) → int?
- calculateConfidence(TransactionEvent event) → double
- isValidTransaction(TransactionEvent event) → bool

#### PATTERN MATCHING:

##### Price Inquiry Patterns:

- "how much be this"
- "what's the price"
- "sɛn na ɛyɛ" (Twi: how much is it)
- "how much you dey sell"
- "price for"

##### Price Response Patterns:

- "\d+ cedis"
- "cost \d+"
- "ɛyɛ cedis \d+"
- "make you give me \d+"

##### Fish Type Patterns:

- "tilapia", "tuo", "apateshi"
- "tuna", "light meat"
- "mackerel", "kpanla", "titus"
- "sardine", "herring"
- "red fish", "snapper"

##### Payment Confirmation Patterns:

- "here be your money"
- "take your \d+ cedis"
- "sika ni" (Twi: here's the money)
- "thank you"
- "medaase" (Twi: thank you)

#### CONFIDENCE SCORING FACTORS:

1. Clear price mentioned: +0.3
2. Clear fish type: +0.2
3. Payment confirmation: +0.3
4. Complete conversation flow: +0.2
5. Speaker identification: +0.1

#### THRESHOLD:

- Auto-record if confidence > 0.8
- Flag for review if 0.5 < confidence <= 0.8
- Ignore if confidence < 0.5

#### FALSE POSITIVE FILTERING:

- Ignore if discussing weather/family
- Ignore if numbers without context
- Ignore if talking about other markets
- Ignore if conversation too short (<5 seconds)

#### DATA CLASSES:

```
class TransactionEvent {
    String id;
    DateTime timestamp;
```

```

TransactionState state;
String? fishType;
double? price;
int? quantity;
double confidence;
List<String> rawTranscripts;
String? customer;
String? seller;
}

```

#### EXAMPLE USAGE:

```

final detector = TransactionDetectionService();
final transcript = "How much be this tilapia?";
final event = detector.processTranscript(transcript, "customer");
if (event != null && event.confidence > 0.8) {
  // Save transaction
}

```

Generate the complete TransactionDetectionService with all pattern matching logic, state machine, and confidence scoring implemented.

...

---

#### ### Prompt 6: Firebase Service

...

Create a Firebase Service (lib/services/firebase\_service.dart) to handle all Firebase operations:

CLASS: FirebaseService

#### FEATURES:

1. User authentication (anonymous for MVP)
2. Transaction CRUD operations
3. Real-time transaction syncing
4. Daily summary calculations
5. Offline persistence
6. Data backup and recovery

#### KEY METHODS:

##### AUTHENTICATION:

- signInAnonymously() → Future<String> userId
- getCurrentUserId() → String?
- createUserProfile(UserProfile profile) → Future<void>

##### TRANSACTION OPERATIONS:

- saveTransaction(Transaction transaction) → Future<String> transactionId
- getTransaction(String id) → Future<Transaction?>
- updateTransaction(String id, Transaction transaction) → Future<void>
- deleteTransaction(String id) → Future<void>
- getTodayTransactions() → Stream<List<Transaction>>
- getTransactionsByDateRange(DateTime start, DateTime end) → Future<List<Transaction>>

##### SUMMARY OPERATIONS:

- getDailySummary(DateTime date) → Future<DailySummary>



- getWeeklySummary(DateTime weekStart) → Future<WeeklySummary>
- getMonthlySummary(int year, int month) → Future<MonthlySummary>

#### DATA CLASSES:

```
class DailySummary {  
    DateTime date;  
    double totalSales;  
    int transactionCount;  
    String topFishType;  
    double averageTransaction;  
    Map<String, int> fishTypeCounts;  
    Map<String, double> fishTypeTotals;  
}
```

```
class WeeklySummary {  
    DateTime weekStart;  
    double totalSales;  
    int transactionCount;  
    List<DailySummary> dailySummaries;  
    Map<String, double> trends;  
}
```

#### FIRESTORE STRUCTURE:

```
users/{userId}/  
- profile (document)  
- transactions/{transactionId} (collection)  
  - timestamp  
  - fishType  
  - amount  
  - quantity  
  - confidence  
  - status  
  - rawTranscript  
- daily_summaries/{date} (collection)  
  - computed daily totals  
- settings (document)
```

#### OPTIMIZATION:

- Use Firestore batch writes for multiple operations
- Implement pagination for large transaction lists
- Cache frequently accessed data
- Use Firestore indexes for common queries
- Compress large text fields

#### OFFLINE SUPPORT:

- Enable offline persistence
- Queue writes when offline
- Sync when connection restored
- Handle conflict resolution

#### SECURITY:

- Validate data before writes
- Sanitize user inputs
- Implement proper Firestore security rules
- Handle authentication errors gracefully

### ERROR HANDLING:

- Network errors
- Permission denied
- Quota exceeded
- Invalid data formats
- Concurrent modification

EXAMPLE USAGE:

```
final firebaseService = FirebaseService();
await firebaseService.signInAnonymously();
await firebaseService.saveTransaction(transaction);
firebaseService.getTodayTransactions().listen((transactions) {
  // Update UI
});
```

Generate the complete `FirestoreService` with all CRUD operations, summaries, and proper error handling.

...

—

## ## UI COMPONENT PROMPTS

### ### Prompt 7: Home Screen

...

Create the main Home Screen (`lib/screens/home_screen.dart`) with the following requirements:

### DESIGN PHILOSOPHY:

- Large, sun-readable text
- Simple, intuitive controls
- Low-literacy friendly
- One-handed operation
- Clear visual feedback


LAYOUT STRUCTURE:

## FEATURES:


1. Start/Stop listening button (large, centered)
2. Real-time sales total (updates live)
3. Transaction count for today
4. Last 3 transactions preview
5. Visual listening indicator (animated when active)
6. Quick manual entry button (floating action button)

## BUTTON STATES:

### START LISTENING:

- Background: Green (#4CAF50)
- Icon:  Microphone
- Text: "Start Listening"
- Size: 200x200 dp (very large)

### STOP LISTENING:

- Background: Red (#F44336)
- Icon:  Stop
- Text: "Stop Listening"
- Pulsing animation when active

## LISTENING INDICATOR:

- Small pulsing dot in corner
- Shows "Listening..." text
- Audio wave animation (optional)

## TODAY'S SUMMARY CARD:

- Large currency symbol (GH¢)
- Extra large number (48sp font)
- Green color if profitable
- Show comparison to yesterday (optional)

## STATE MANAGEMENT:

- Use Provider for state management
- Real-time updates from Firestore
- Handle loading states
- Error state displays
- Empty state (no transactions yet)

## INTERACTIONS:

- Tap start/stop button → toggle listening
- Tap transaction → view details
- Long press transaction → edit/delete
- Swipe to refresh transaction list
- Pull to see more details

## PERMISSIONS HANDLING:

- Check microphone permission on start
- Show permission explanation if denied
- Provide "Go to Settings" option

## ACCESSIBILITY:

- Screen reader support
- Large touch targets (min 48dp)

- High contrast colors
- Clear focus indicators

#### CODE STRUCTURE:

```
class HomeScreen extends StatefulWidget {  
  // Use Provider to access:  
  // - AudioService  
  // - FirebaseService  
  // - TransactionDetectionService  
}
```

```
class _HomeScreenState extends State<HomeScreen> {  
  bool isListening = false;
```

```
  @override  
  void initState() {  
    // Initialize services  
    // Set up listeners  
  }
```

```
  Future<void> _toggleListening() async {  
    // Start/stop audio recording  
    // Update UI state  
  }
```

```
  Widget _buildStartStopButton() {  
    // Large circular button  
  }
```

```
  Widget _buildTodaySummary() {  
    // Sales total card  
  }
```

```
  Widget _buildRecentTransactions() {  
    // Transaction list preview  
  }  
}
```

#### EXAMPLE BEHAVIORS:

1. User taps "Start Listening":
  - Request microphone permission if needed
  - Start AudioService
  - Show listening indicator
  - Button changes to "Stop Listening"
2. Transaction detected:
  - Play subtle sound (optional)
  - Update sales total with animation
  - Add to recent transactions list
  - Show brief success indicator
3. User taps transaction:
  - Navigate to transaction details
  - Allow edit/delete actions

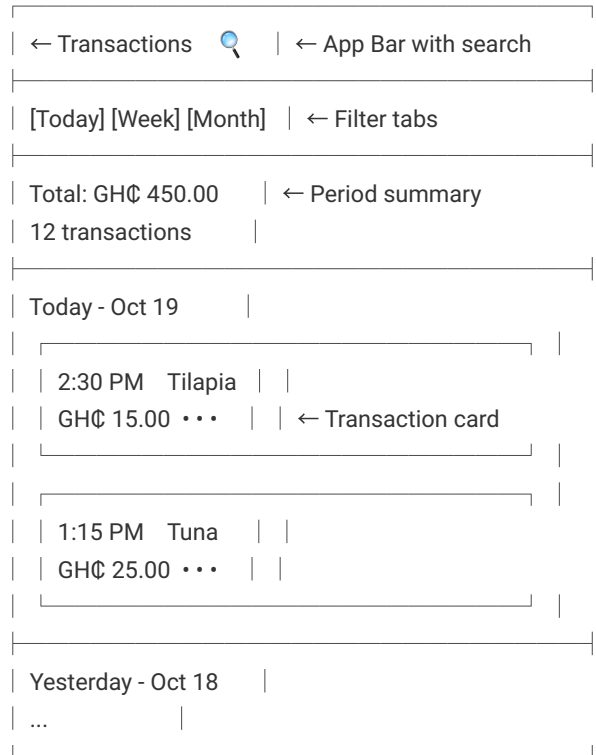
Generate the complete HomeScreen with all features, proper state management, and error handling.

### Prompt 8: Transaction List Screen

Create the Transaction List Screen (lib/screens/transaction\_list\_screen.dart):

PURPOSE: Display all transactions with filtering and search capabilities

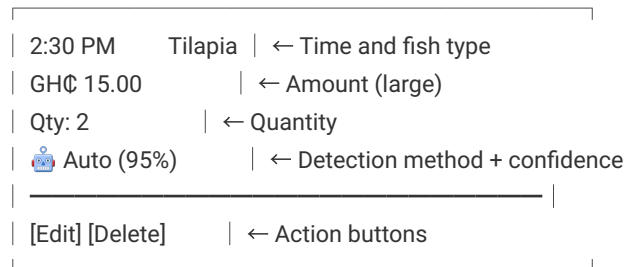
LAYOUT:



FEATURES:

1. Filter by date range (today, week, month, custom)
2. Search transactions by fish type
3. Group transactions by date
4. Show daily subtotals
5. Swipe actions (edit, delete)
6. Pull to refresh
7. Pagination for large lists

TRANSACTION CARD DESIGN:



CARD VARIATIONS:

- Auto detected: Green border, robot icon
- Manual entry: Blue border, hand icon

- Edited: Orange border, pencil icon
- Low confidence: Yellow border, warning icon

#### FILTERING:

- Today: Show all today's transactions
- Week: Last 7 days grouped by date
- Month: Current month grouped by date
- Custom: Date range picker

#### SEARCH:

- Search by fish type
- Filter by price range
- Filter by confidence level
- Filter by detection method

#### SORTING OPTIONS:

- Time (newest first - default)
- Amount (highest first)
- Fish type (alphabetical)
- Confidence (highest first)

#### ACTIONS:

1. Tap transaction → View details
2. Swipe left → Delete (with confirmation)
3. Swipe right → Edit
4. Long press → Select multiple
5. Pull down → Refresh

#### EMPTY STATES:

- No transactions today: "Start listening to record sales"
- No results for search: "No transactions found for '{query}'"
- No transactions in range: "No sales recorded in this period"

#### STATE MANAGEMENT:

```
class TransactionListScreen extends StatefulWidget {  
  final DateRange? initialRange;  
}  
  
class _TransactionListScreenState extends State<TransactionListScreen> {  
  DateRange selectedRange = DateRange.today;  
  String searchQuery = "";  
  List<Transaction> filteredTransactions = [];  
  
  Future<void> _loadTransactions() async {  
    // Load from Firebase  
  }  
  
  void _filterTransactions() {  
    // Apply filters and search  
  }  
  
  Future<void> _deleteTransaction(String id) async {  
    // Confirm and delete  
  }  
}
```

OPTIMIZATIONS:

- Lazy loading (load 20 transactions at a time)
- Cache loaded transactions
- Virtualized scrolling for large lists
- Debounced search input
- Batch delete operations

Generate the complete TransactionListScreen with filtering, search, and all interactions.

...

--

### Prompt 9: Settings Screen

...

Create the Settings Screen (lib/screens/settings\_screen.dart):

PURPOSE: Allow users to configure app behavior and preferences

SETTINGS CATEGORIES:

1. LISTENING SETTINGS

Auto Start Listening	← Toggle
Start when app opens	

Smart Listening	← Toggle
Only during market hours	

Market Hours	
6:00 AM - 6:00 PM	← Time picker

2. LANGUAGE SETTINGS

Primary Language	
English ▼	← Dropdown

Secondary Language	
Twi ▼	

3. TRANSACTION SETTINGS

Auto-save Threshold	
80% confidence	← Slider (50-95%)

Manual Review Required	← Toggle
Review low confidence	

Daily Summary Time	
6:00 PM	← Time picker

4. FISH TYPES


--	--

Common Fish Types	
<input checked="" type="checkbox"/> Tilapia	
<input checked="" type="checkbox"/> Tuna	
<input checked="" type="checkbox"/> Mackerel	
 Sardines	
[+ Add Custom]	

## 5. NOTIFICATIONS

Daily Summary	← Toggle
End of day report	
Transaction Alerts	← Toggle
Notify on each sale	
Low Confidence Alert	← Toggle
Review needed	

## 6. DATA & BACKUP

Cloud Backup	← Toggle
Auto backup to Firebase	
[Export Data]	← Button
Download CSV	
[Clear All Data]	← Dangerous action
 Cannot be undone	

## 7. ACCOUNT & PROFILE

Business Name	
[Fish Market Stall]	← Text input
Phone Number	
[+233 XX XXX XXXX]	
Market Location	
[Kaneshie Market]	

## 8. ABOUT & SUPPORT

App Version	
1.0.0 (MVP)	
[Help & Tutorial]	
[Report a Problem]	
[Rate the App]	

## KEY FEATURES:

- All settings saved to Firestore



- Sync across devices (if implemented)
- Validate inputs before saving
- Confirm dangerous actions (clear data)
- Show success/error messages
- Restore defaults option

#### DATA PERSISTENCE:


```
class AppSettings {
    bool autoStartListening;
    bool smartListening;
    TimeOfDay marketStartTime;
    TimeOfDay marketEndTime;
    String primaryLanguage;
    String secondaryLanguage;
    double autoSaveThreshold;
    bool requireManualReview;
    TimeOfDay dailySummaryTime;
    List<String> commonFishTypes;
    bool dailySummaryNotification;
    bool transactionAlerts;
    bool lowConfidenceAlert;
    bool cloudBackup;
    String businessName;
    String phoneNumber;
    String marketLocation;
}
```

#### INTERACTIONS:

- Toggle switches update immediately
- Text inputs save on focus loss
- Sliders show value as dragging
- Time/date pickers open dialogs
- Dangerous actions require confirmation

#### EXAMPLE CONFIRMATIONS:

"Clear All Data":

 Delete All Data?

This will permanently  
delete all transactions  
and cannot be undone.

[Cancel] [Delete]

"Export Data":

- Generate CSV file
- Show "Export successful" message
- Provide share options

#### CODE STRUCTURE:

```
class SettingsScreen extends StatefulWidget {}

class _SettingsScreenState extends State<SettingsScreen> {
    late AppSettings settings;
```

```
@override
void initState() {
  _loadSettings();
}

Future<void> _loadSettings() async {
  // Load from SharedPreferences and Firestore
}

Future<void> _saveSettings() async {
  // Save to SharedPreferences and Firestore
}

Widget _buildSection(String title, List<Widget> children) {
  // Reusable section builder
}

Widget _buildToggleSetting(String title, String subtitle, bool value, Function(bool) onChanged) {
  // Toggle switch setting
}

Widget _buildSliderSetting(String title, double value, double min, double max, Function(double)
onChanged) {
  // Slider setting
}
}
```

Generate the complete SettingsScreen with all features and proper persistence.  
...

---

### Prompt 10: Onboarding Screen

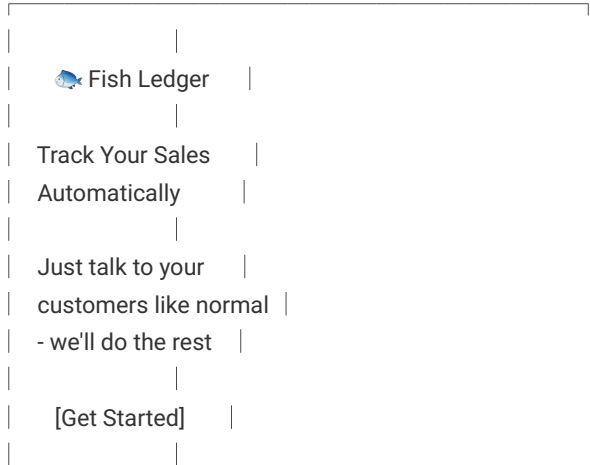
...

Create an Onboarding Screen (lib/screens/onboarding\_screen.dart) for first-time users:

PURPOSE: Guide new users through setup and explain app functionality

ONBOARDING FLOW (4 Screens):

SCREEN 1: WELCOME



SCREEN 2: HOW IT WORKS



We Listen Quietly

1. Customer asks price


2. You tell the price

3. Sale is confirmed

4. We record it!

[Next]

SCREEN 3: PRIVACY



Your Privacy Matters

• Only you see data

• No audio saved

• Works offline

• Delete anytime

[Next]

SCREEN 4: SETUP

Quick Setup

Business Name:

Market Location:

Main Fish You Sell:

☒ Tilapia

☒ Tuna

☐ Mackerel

☐ Sardines

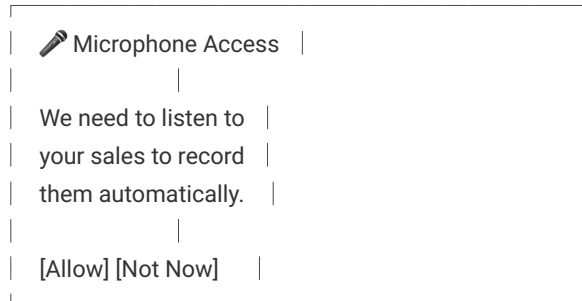
[Start Using App]

#### FEATURES:

- Swipe to navigate between screens
- Skip button (top right)
- Animated transitions
- Simple illustrations or icons
- Large, readable text
- Show only once (unless user resets)

#### PERMISSIONS REQUEST:

After onboarding, request permissions:



#### CODE STRUCTURE:

```
class OnboardingScreen extends StatefulWidget {}
```

```
class _OnboardingScreenState extends State<OnboardingScreen> {
```

```
  final PageController _pageController = PageController();
```

```
  int _currentPage = 0;
```

```
  final TextEditingController _businessNameController = TextEditingController();
```

```
  final TextEditingController _locationController = TextEditingController();
```

```
  List<String> _selectedFishTypes = [];
```

```
  void _nextPage() {
```

```
    if (_currentPage < 3) {
```

```
      _pageController.nextPage(  
        duration: Duration(milliseconds: 300),  
        curve: Curves.easeInOut,  
      );
```

```
    } else {
```

```
      _completeOnboarding();
```

```
    }
```

```
  }
```

```
  Future<void> _completeOnboarding() async {
```

```
    // Save user profile
```

```
    // Mark onboarding as completed
```

```
    // Request permissions
```

```
    // Navigate to home screen
```

```
  }
```

```
  Widget _buildPage1() { /* Welcome */ }
```

```
  Widget _buildPage2() { /* How it works */ }
```

```
  Widget _buildPage3() { /* Privacy */ }
```

```
  Widget _buildPage4() { /* Setup */ }
```

```
}
```

#### VALIDATION:

- Business name: Optional but recommended
- Location: Optional
- Fish types: At least one must be selected
- Show error if setup incomplete

#### PERSISTENCE:

- Save onboarding completion to SharedPreferences
- Save user profile to Firestore
- Don't show again unless user explicitly requests

Generate the complete OnboardingScreen with all pages and smooth navigation.

```
...
```

```
---
```

### ## INTEGRATION & TESTING PROMPTS

#### ### Prompt 11: Main App Integration

```
...
```

Create the main application entry point (lib/main.dart) that integrates all services and screens:

#### REQUIREMENTS:

1. Initialize Firebase
2. Set up service providers
3. Configure app theme
4. Handle routing
5. Check onboarding status
6. Request permissions
7. Set up background services

#### MAIN.DART STRUCTURE:

```
void main() async {  
  WidgetsFlutterBinding.ensureInitialized();  
  
  // Initialize Firebase  
  await Firebase.initializeApp();  
  
  // Initialize services  
  final audioService = AudioService();  
  final speechService = SpeechRecognitionService();  
  final detectionService = TransactionDetectionService();  
  final firebaseService = FirebaseService();  
  
  // Sign in anonymously  
  await firebaseService.signInAnonymously();  
  
  runApp(  
    MultiProvider(  
      providers: [  
        Provider<AudioService>.value(value: audioService),  
        Provider<SpeechRecognitionService>.value(value: speechService),
```

```

        Provider<TransactionDetectionService>.value(value: detectionService),
        Provider<FirebaseService>.value(value: firebaseService),
        ChangeNotifierProvider<AppState>(create: (_) => AppState()),
    ],
    child: FishLedgerApp(),
  ),
);
}

```

```

class FishLedgerApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Fish Ledger',
      theme: _buildAppTheme(),
      home: _determineInitialScreen(),
      routes: {
        '/home': (context) => HomeScreen(),
        '/transactions': (context) => TransactionListScreen(),
        '/settings': (context) => SettingsScreen(),
        '/onboarding': (context) => OnboardingScreen(),
      },
    );
  }
}

```

```

ThemeData _buildAppTheme() {
  return ThemeData(
    primarySwatch: Colors.blue,
    brightness: Brightness.light,

    // Large, readable text
    textTheme: TextTheme(
      displayLarge: TextStyle(fontSize: 48, fontWeight: FontWeight.bold),
      displayMedium: TextStyle(fontSize: 36, fontWeight: FontWeight.bold),
      bodyLarge: TextStyle(fontSize: 18),
      bodyMedium: TextStyle(fontSize: 16),
    ),

    // Large buttons
    elevatedButtonTheme: ElevatedButtonThemeData(
      style: ElevatedButton.styleFrom(
        minimumSize: Size(200, 60),
        textStyle: TextStyle(fontSize: 20),
      ),
    ),

    // High contrast colors
    colorScheme: ColorScheme.fromSeed(
      seedColor: Colors.blue,
      brightness: Brightness.light,
    ),
  );
}

```

```

Widget _determineInitialScreen() {
  // Check if onboarding completed
}

```

```

return FutureBuilder<bool>(  
    future: _checkOnboardingStatus(),  
    builder: (context, snapshot) {  
        if (!snapshot.hasData) {  
            return SplashScreen();  
        }  
  
        if (snapshot.data == true) {  
            return HomeScreen();  
        } else {  
            return OnboardingScreen();  
        }  
    },  
);  
}

```

```

Future<bool> _checkOnboardingStatus() async {  
    final prefs = await SharedPreferences.getInstance();  
    return prefs.getBool('onboarding_completed') ?? false;  
}  
}

```

#### APP STATE MANAGEMENT:

```

class AppState extends ChangeNotifier {  
    bool isListening = false;  
    List<Transaction> todayTransactions = [];  
    double todayTotal = 0.0;  
  
    void startListening() {  
        isListening = true;  
        notifyListeners();  
    }  
  
    void stopListening() {  
        isListening = false;  
        notifyListeners();  
    }  
  
    void addTransaction(Transaction transaction) {  
        todayTransactions.add(transaction);  
        todayTotal += transaction.amount;  
        notifyListeners();  
    }  
  
    void updateTransaction(Transaction transaction) {  
        final index = todayTransactions.indexWhere((t) => t.id == transaction.id);  
        if (index != -1) {  
            todayTransactions[index] = transaction;  
            _recalculateTotal();  
            notifyListeners();  
        }  
    }  
  
    void deleteTransaction(String id) {  
        todayTransactions.removeWhere((t) => t.id == id);  
        _recalculateTotal();  
    }  
}

```

```

        notifyListeners();
    }

    void _recalculateTotal() {
        todayTotal = todayTransactions.fold(0.0, (sum, t) => sum + t.amount);
    }
}

```

#### SPLASH SCREEN:

```

class SplashScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(Icons.mic, size: 100, color: Colors.blue),
            SizedBox(height: 20),
            Text('Fish Ledger', style: Theme.of(context).textTheme.displayMedium),
            SizedBox(height: 20),
            CircularProgressIndicator(),
          ],
        ),
      ),
    );
  }
}

```

#### BACKGROUND PROCESSING:

Set up background task to process audio continuously:

```

class BackgroundAudioProcessor {
  static Future<void> initialize() async {
    // Register background task
    // Process audio chunks
    // Detect transactions
    // Save to database
  }

  static Future<void> processAudioChunk(Uint8List audioData) async {
    // Get speech recognition
    final transcript = await speechService.recognizeAudio(audioData);

    // Detect transaction
    final transaction = await detectionService.processTranscript(transcript);

    // Save if valid
    if (transaction != null && transaction.confidence > 0.8) {
      await firebaseService.saveTransaction(transaction);
    }
  }
}

```

#### PERMISSIONS:

Handle runtime permissions properly:



```
class PermissionManager {
  static Future<bool> requestMicrophonePermission() async {
    final status = await Permission.microphone.request();
    return status.isGranted;
  }
}
```

```
static Future<bool> checkMicrophonePermission() async {
  final status = await Permission.microphone.status;
  return status.isGranted;
}
```

```
static Future<void> openAppSettings() async {
  await openAppSettings();
}
}
```

ERROR HANDLING:  
Global error handler:

```
class GlobalErrorHandler {
  static void initialize() {
    FlutterError.onError = (FlutterErrorDetails details) {
      // Log to Firebase Crashlytics (if enabled)
      print('Flutter Error: ${details.exception}');
    };
  }
}
```

```
static void handleError(dynamic error, StackTrace? stackTrace) {
  // Log error
  print('Error: $error');
  if (stackTrace != null) {
    print('Stack trace: $stackTrace');
  }
}
}
```

Generate the complete main.dart with all initialization, routing, state management, and error handling.  
...

---

### Prompt 12: Core Processing Loop

...

Create the core processing loop that connects audio recording → speech recognition → transaction detection → database storage:

FILE: lib/services/processing\_pipeline.dart

CLASS: ProcessingPipeline

PURPOSE: Orchestrate the complete transaction detection workflow

WORKFLOW:

1. Audio Service continuously records

2. Audio chunks sent to Speech Recognition
3. Transcripts analyzed by Transaction Detection
4. Valid transactions saved to Firebase
5. UI updated in real-time

#### KEY METHODS:

- start() → Begin processing pipeline
- stop() → Stop and cleanup
- pause() → Temporarily pause
- resume() → Resume processing
- getStatus() → PipelineStatus

#### IMPLEMENTATION:

```
class ProcessingPipeline {
    final AudioService _audioService;
    final SpeechRecognitionService _speechService;
    final TransactionDetectionService _detectionService;
    final FirebaseService _firebaseService;

    StreamSubscription? _audioSubscription;
    bool _isRunning = false;

    // Transaction state machine
    TransactionStateMachine _stateMachine = TransactionStateMachine();

    // Buffer for recent transcripts
    List<TranscriptSegment> _recentTranscripts = [];

    ProcessingPipeline({
        required AudioService audioService,
        required SpeechRecognitionService speechService,
        required TransactionDetectionService detectionService,
        required FirebaseService firebaseService,
    }) : _audioService = audioService,
        _speechService = speechService,
        _detectionService = detectionService,
        _firebaseService = firebaseService;

    Future<void> start() async {
        if (_isRunning) return;

        print('Starting processing pipeline...');

        // Start audio recording
        await _audioService.startListening();

        // Listen to audio stream
        _audioSubscription = _audioService.getAudioStream().listen(
            _processAudioChunk,
            onError: _handleError,
        );

        _isRunning = true;
    }
}
```

```

Future<void> _processAudioChunk(Uint8List audioData) async {
  try {
    // Skip if no voice detected
    if (!_audioService.isVoiceDetected()) {
      return;
    }

    // Step 1: Convert audio to text
    final recognitionResult = await _speechService.recognizeAudio(audioData);

    if (recognitionResult.transcript.isEmpty) {
      return;
    }

    print('Transcript: ${recognitionResult.transcript}');

    // Step 2: Add to recent transcripts buffer
    _recentTranscripts.add(TranscriptSegment(
      text: recognitionResult.transcript,
      timestamp: DateTime.now(),
      confidence: recognitionResult.confidence,
    ));

    // Keep only last 10 transcripts (rolling window)
    if (_recentTranscripts.length > 10) {
      _recentTranscripts.removeAt(0);
    }

    // Step 3: Analyze for transaction patterns
    final transactionEvent = _detectionService.processTranscript(
      recognitionResult.transcript,
      'unknown', // Speaker identification comes later
    );

    // Step 4: Update state machine
    if (transactionEvent != null) {
      _stateMachine.processEvent(transactionEvent);

      // Check if transaction is complete
      if (_stateMachine.isTransactionComplete()) {
        await _saveTransaction(_stateMachine.getCurrentTransaction());
        _stateMachine.reset();
      }
    }
  } catch (e, stackTrace) {
    _handleError(e, stackTrace);
  }
}

```

```

Future<void> _saveTransaction(TransactionEvent event) async {
  // Convert event to transaction
  final transaction = Transaction(
    id: DateTime.now().millisecondsSinceEpoch.toString(),
    timestamp: DateTime.now(),
    fishType: event.fishType ?? 'unknown',
  );
}

```

```

        amount: event.price ?? 0.0,
        quantity: event.quantity ?? 1,
        confidence: event.confidence,
        rawTranscript: event.rawTranscripts.join(' | '),
        status: TransactionStatus.auto,
        sellerId: _firebaseService.getCurrentUserId() ?? 'anonymous',
    );

    // Save to Firebase
    try {
        await _firebaseService.saveTransaction(transaction);
        print('Transaction saved: ${transaction.fishType} - GH¢${transaction.amount}');

        // Notify UI (through state management)
        _notifyTransactionDetected(transaction);

    } catch (e) {
        print('Error saving transaction: $e');
        // Queue for retry
        _queueFailedTransaction(transaction);
    }
}

void _notifyTransactionDetected(Transaction transaction) {
    // This would be handled by your state management solution
    // For example, using Provider or Riverpod
    // AppState.instance.addTransaction(transaction);
}

void _queueFailedTransaction(Transaction transaction) {
    // Store locally and retry later
    // This ensures no data loss during network issues
}

void _handleError(dynamic error, [StackTrace? stackTrace]) {
    print('Pipeline error: $error');
    if (stackTrace != null) {
        print('Stack trace: $stackTrace');
    }

    // Log to analytics/crashlytics
    // Continue processing (don't crash)
}

Future<void> stop() async {
    if (!_isRunning) return;

    print('Stopping processing pipeline...');

    // Cancel audio subscription
    await _audioSubscription?.cancel();

    // Stop audio recording
    await _audioService.stopListening();

    // Reset state

```

```

_stateMachine.reset();
_recentTranscripts.clear();

_isRunning = false;
}

Future<void> pause() async {
  await _audioService.pauseListening();
}

Future<void> resume() async {
  await _audioService.resumeListening();
}

PipelineStatus getStatus() {
  return PipelineStatus(
    isRunning: _isRunning,
    isListening: _audioService.isVoiceDetected(),
    currentState: _stateMachine.currentState,
    recentTranscripts: _recentTranscripts,
  );
}
}

```

#### SUPPORTING CLASSES:

```

class TranscriptSegment {
  final String text;
  final DateTime timestamp;
  final double confidence;

  TranscriptSegment({
    required this.text,
    required this.timestamp,
    required this.confidence,
  });
}

class TransactionStateMachine {
  TransactionState currentState = TransactionState.idle;
  TransactionEvent? _currentTransaction;
  DateTime? _lastEventTime;

  final Duration _timeoutDuration = Duration(minutes: 2);

  void processEvent(TransactionEvent event) {
    _lastEventTime = DateTime.now();

    switch (currentState) {
      case TransactionState.idle:
        if (event.type == EventType.pricelInquiry) {
          currentState = TransactionState.pricelInquiry;
          _currentTransaction = event;
        }
        break;
    }
  }
}

```

```

case TransactionState.priceInquiry:
    if (event.type == EventType.priceResponse) {
        currentState = TransactionState.priceGiven;
        _mergeEvent(event);
    }
    break;

case TransactionState.priceGiven:
    if (event.type == EventType.negotiation) {
        currentState = TransactionState.negotiation;
        _mergeEvent(event);
    } else if (event.type == EventType.payment) {
        currentState = TransactionState.payment;
        _mergeEvent(event);
    }
    break;

case TransactionState.negotiation:
    if (event.type == EventType.payment) {
        currentState = TransactionState.payment;
        _mergeEvent(event);
    }
    break;

case TransactionState.payment:
    if (event.type == EventType.confirmation) {
        currentState = TransactionState.completed;
        _mergeEvent(event);
    }
    break;

case TransactionState.completed:
    // Transaction done
    break;
}

// Check for timeout
_checkTimeout();
}

void _mergeEvent(TransactionEvent event) {
    // Merge new event data into current transaction
    if (_currentTransaction != null) {
        _currentTransaction = _currentTransaction!.copyWith(
            fishType: event.fishType ?? _currentTransaction!.fishType,
            price: event.price ?? _currentTransaction!.price,
            quantity: event.quantity ?? _currentTransaction!.quantity,
            rawTranscripts: [..._currentTransaction!.rawTranscripts, ...event.rawTranscripts],
        );
    }
}

void _checkTimeout() {
    if (_lastEventTime != null) {
        final elapsed = DateTime.now().difference(_lastEventTime!);
        if (elapsed > _timeoutDuration) {

```

```

        reset(); // Timeout - abandon transaction
    }
}

bool isTransactionComplete() {
    return currentState == TransactionState.completed &&
        _currentTransaction != null &&
        _currentTransaction!.price != null;
}

TransactionEvent getCurrentTransaction() {
    return _currentTransaction!;
}

void reset() {
    currentState = TransactionState.idle;
    _currentTransaction = null;
    _lastEventTime = null;
}
}

```

```

class PipelineStatus {
    final bool isRunning;
    final bool isListening;
    final TransactionState currentState;
    final List<TranscriptSegment> recentTranscripts;

```

```

    PipelineStatus({
        required this.isRunning,
        required this.isListening,
        required this.currentState,
        required this.recentTranscripts,
    });
}

```

USAGE IN HOME SCREEN:

```

class _HomeScreenState extends State<HomeScreen> {
    late ProcessingPipeline _pipeline;

    @override
    void initState() {
        super.initState();

        _pipeline = ProcessingPipeline(
            audioService: context.read<AudioService>(),
            speechService: context.read<SpeechRecognitionService>(),
            detectionService: context.read<TransactionDetectionService>(),
            firebaseService: context.read<FirebaseService>(),
        );
    }

    Future<void> _toggleListening() async {
        if (_pipeline.getStatus().isRunning) {
            await _pipeline.stop();

```

```

    } else {
      await _pipeline.start();
    }
    setState(() {});
  }
}

```

Generate the complete ProcessingPipeline with state machine, error handling, and integration points.

...

--

### ### Prompt 13: Testing & Debugging Tools

...

Create comprehensive testing and debugging utilities:

FILE: lib/utils/debug\_tools.dart

PURPOSE: Tools for testing and debugging the fish market transaction detection

FEATURES:

1. Mock audio generator (simulate market conversations)
2. Transaction accuracy tester
3. Pattern matching validator
4. Performance monitor
5. Debug UI overlay

MOCK AUDIO GENERATOR:

```

class MockAudioGenerator {
  // Generate synthetic market conversations for testing

  static List<String> generateMockConversations() {
    return [
      // Price inquiry patterns
      "How much be this tilapia?",
      "Sen na eye? This fish",
      "What's the price of the tuna?",

      // Price response patterns
      "This one be 15 cedis",
      "Eye cedis dunum", // It's 10 cedis
      "I dey sell am 20 Ghana cedis",

      // Negotiation patterns
      "Too much o, reduce small",
      "Can you do 12 cedis?",
      "Eye dooso", // It's too much

      // Payment patterns
      "Here be your money",
      "Take your 15 cedis",
      "Sika ni", // Here's the money

      // Confirmation patterns

```



```

        "Thank you",
        "Medaase",
        "Go well",
    ];
}

static Future<List<MockTransaction>> generateTestScenarios() async {
    return [
        MockTransaction(
            conversation: [
                MockUtterance(speaker: "customer", text: "How much be this tilapia?"),
                MockUtterance(speaker: "seller", text: "15 cedis"),
                MockUtterance(speaker: "customer", text: "Here be your money"),
                MockUtterance(speaker: "seller", text: "Thank you"),
            ],
            expectedResult: Transaction(
                fishType: "tilapia",
                amount: 15.0,
                quantity: 1,
            ),
        ),

        MockTransaction(
            conversation: [
                MockUtterance(speaker: "customer", text: "Sɛn na ɛyɛ?"),
                MockUtterance(speaker: "seller", text: "Ɛyɛ cedis aduonum"), // 20 cedis
                MockUtterance(speaker: "customer", text: "Too much, do 18"),
                MockUtterance(speaker: "seller", text: "Okay, 18 cedis"),
                MockUtterance(speaker: "customer", text: "Sika ni"),
            ],
            expectedResult: Transaction(
                fishType: "unknown", // Not mentioned
                amount: 18.0,
                quantity: 1,
            ),
        ),

        // Add 20+ more test scenarios
    ];
}

class MockTransaction {
    final List<MockUtterance> conversation;
    final Transaction expectedResult;

    MockTransaction({
        required this.conversation,
        required this.expectedResult,
    });
}

class MockUtterance {
    final String speaker;
    final String text;

```

```
MockUtterance({required this.speaker, required this.text});
}
```

ACCURACY TESTER:

```
class AccuracyTester {
  final TransactionDetectionService detectionService;

  AccuracyTester(this.detectionService);

  Future<TestResults> runAccuracyTest() async {
    final testScenarios = await MockAudioGenerator.generateTestScenarios();

    int correctDetections = 0;
    int falsePositives = 0;
    int falseNegatives = 0;
    List<TestFailure> failures = [];

    for (final scenario in testScenarios) {
      // Process each conversation
      TransactionEvent? detected;

      for (final utterance in scenario.conversation) {
        final event = detectionService.processTranscript(
          utterance.text,
          utterance.speaker,
        );

        if (event != null) {
          detected = event;
        }
      }

      // Compare detected vs expected
      if (detected != null && scenario.expectedResult != null) {
        if (_transactionsMatch(detected, scenario.expectedResult)) {
          correctDetections++;
        } else {
          failures.add(TestFailure(
            scenario: scenario,
            detected: detected,
            expected: scenario.expectedResult,
            reason: 'Mismatch',
          ));
        }
      } else if (detected != null && scenario.expectedResult == null) {
        falsePositives++;
      } else if (detected == null && scenario.expectedResult != null) {
        falseNegatives++;
      }
    }

    return TestResults(
      totalTests: testScenarios.length,
      correctDetections: correctDetections,
      falsePositives: falsePositives,
```

```

        falseNegatives: falseNegatives,
        accuracy: correctDetections / testScenarios.length,
        failures: failures,
    );
}

bool _transactionsMatch(TransactionEvent detected, Transaction expected) {
    // Allow small price variance
    final priceMatch = (detected.price != null && expected.amount != null) &&
        (detected.price! - expected.amount!).abs() < 0.01;

    // Fish type match (if mentioned)
    final fishMatch = detected.fishType == null ||
        expected.fishType == null ||
        detected.fishType == expected.fishType;

    return priceMatch && fishMatch;
}

}

class TestResults {
    final int totalTests;
    final int correctDetections;
    final int falsePositives;
    final int falseNegatives;
    final double accuracy;
    final List<TestFailure> failures;

    TestResults({
        required this.totalTests,
        required this.correctDetections,
        required this.falsePositives,
        required this.falseNegatives,
        required this.accuracy,
        required this.failures,
    });

    @override
    String toString() {
        return "
Test Results:
-----
Total Tests: $totalTests
Correct: $correctDetections
False Positives: $falsePositives
False Negatives: $falseNegatives
Accuracy: ${((accuracy * 100).toStringAsFixed(1))}%

Failed Tests: ${failures.length}
",
    }
}

class TestFailure {
    final MockTransaction scenario;
    final TransactionEvent? detected;

```

```
final Transaction? expected;
final String reason;
```

```
TestFailure({
    required this.scenario,
    required this.detected,
    required this.expected,
    required this.reason,
});
}
```

PERFORMANCE MONITOR:

```
class PerformanceMonitor {
    static final Map<String, List<Duration>> _measurements = {};
    static final Map<String, int> _counters = {};

    static void startMeasurement(String label) {
        _measurements[label] ??= [];
        _measurements[label]!.add(Duration.zero); // Placeholder
    }

    static void endMeasurement(String label, DateTime startTime) {
        final duration = DateTime.now().difference(startTime);
        if (_measurements.containsKey(label)) {
            _measurements[label]!.last = duration;
        }
    }

    static void incrementCounter(String label) {
        _counters[label] = (_counters[label] ?? 0) + 1;
    }

    static PerformanceReport generateReport() {
        final metrics = <String, PerformanceMetric>{};

        _measurements.forEach((label, durations) {
            final validDurations = durations.where((d) => d.inMilliseconds > 0).toList();

            if (validDurations.isNotEmpty) {
                final avg = validDurations.fold<int>(0, (sum, d) => sum + d.inMilliseconds) /
                    validDurations.length;

                final max = validDurations.map((d) => d.inMilliseconds).reduce((a, b) => a > b ? a : b);
                final min = validDurations.map((d) => d.inMilliseconds).reduce((a, b) => a < b ? a : b);

                metrics[label] = PerformanceMetric(
                    label: label,
                    averageMs: avg,
                    maxMs: max.toDouble(),
                    minMs: min.toDouble(),
                    count: validDurations.length,
                );
            }
        });
    }
}
```

```

        return PerformanceReport(
            metrics: metrics,
            counters: _counters,
        );
    }

    static void reset() {
        _measurements.clear();
        _counters.clear();
    }
}

class PerformanceMetric {
    final String label;
    final double averageMs;
    final double maxMs;
    final double minMs;
    final int count;

    PerformanceMetric({
        required this.label,
        required this.averageMs,
        required this.maxMs,
        required this.minMs,
        required this.count,
    });
}

class PerformanceReport {
    final Map<String, PerformanceMetric> metrics;
    final Map<String, int> counters;

    PerformanceReport({
        required this.metrics,
        required this.counters,
    });

    @override
    String toString() {
        final buffer = StringBuffer();
        buffer.writeln('Performance Report:');
        buffer.writeln('=====');

        metrics.forEach((label, metric) {
            buffer.writeln('$label:');
            buffer.writeln(' Avg: ${metric.averageMs.toStringAsFixed(2)}ms');
            buffer.writeln(' Max: ${metric.maxMs.toStringAsFixed(2)}ms');
            buffer.writeln(' Min: ${metric.minMs.toStringAsFixed(2)}ms');
            buffer.writeln(' Count: ${metric.count}');
        });

        buffer.writeln('\nCounters:');
        counters.forEach((label, count) {
            buffer.writeln(' $label: $count');
        });
    }
}

```

```

    return buffer.toString();
  }
}

```

DEBUG UI OVERLAY:

```

class DebugOverlay extends StatelessWidget {
  final Widget child;
  final ProcessingPipeline pipeline;

  const DebugOverlay({
    required this.child,
    required this.pipeline,
  });

  @override
  Widget build(BuildContext context) {
    return Stack(
      children: [
        child,

        // Debug panel (bottom right)
        Positioned(
          bottom: 80,
          right: 10,
          child: Container(
            padding: EdgeInsets.all(8),
            decoration: BoxDecoration(
              color: Colors.black.withOpacity(0.7),
              borderRadius: BorderRadius.circular(8),
            ),
            child: StreamBuilder<PipelineStatus>(
              stream: _statusStream(),
              builder: (context, snapshot) {
                if (!snapshot.hasData) return SizedBox();

                final status = snapshot.data!;
                return Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      'Debug Info',
                      style: TextStyle(
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                    SizedBox(height: 4),
                    Text(
                      'Running: ${status.isRunning}',
                      style: TextStyle(color: Colors.white, fontSize: 10),
                    ),
                    Text(
                      'Listening: ${status.isListening}',
                      style: TextStyle(color: Colors.white, fontSize: 10),
                    ),
                  ],
                );
              },
            ),
          ),
        ),
      ],
    );
  }
}

```

```

        Text(
          'State: ${status.currentState}',
          style: TextStyle(color: Colors.white, fontSize: 10),
        ),
        Text(
          'Transcripts: ${status.recentTranscripts.length}',
          style: TextStyle(color: Colors.white, fontSize: 10),
        ),
      ],
    );
  },
),
),
),
),

// Toggle debug button
Positioned(
  bottom: 20,
  right: 10,
  child: FloatingActionButton(
    mini: true,
    child: Icon(Icons.bug_report),
    onPressed: () => _showDebugPanel(context),
  ),
),
],
);
}

Stream<PipelineStatus> _statusStream() async* {
  while (true) {
    await Future.delayed(Duration(seconds: 1));
    yield pipeline.getStatus();
  }
}

void _showDebugPanel(BuildContext context) {
  showModalBottomSheet(
    context: context,
    builder: (context) => DebugPanel(pipeline: pipeline),
  );
}

class DebugPanel extends StatelessWidget {
  final ProcessingPipeline pipeline;

  const DebugPanel({required this.pipeline});

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.all(16),
      child: ListView(
        children: [
          Text('Debug Panel', style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold)),

```

```

        SizedBox(height: 16),

        ElevatedButton(
          onPressed: () async {
            final tester = AccuracyTester(TransactionDetectionService());
            final results = await tester.runAccuracyTest();
            print(results.toString());
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text('Accuracy: ${(results.accuracy * 100).toStringAsFixed(1)}%')),
            );
          },
          child: Text('Run Accuracy Test'),
        ),

        ElevatedButton(
          onPressed: () {
            final report = PerformanceMonitor.generateReport();
            print(report.toString());
          },
          child: Text('Show Performance Report'),
        ),

        ElevatedButton(
          onPressed: () {
            PerformanceMonitor.reset();
          },
          child: Text('Reset Counters'),
        ),

        SizedBox(height: 16),
        Text('Recent Transcripts:', style: TextStyle(fontWeight: FontWeight.bold)),
        ...pipeline.getStatus().recentTranscripts.map((t) =>
          ListTile(
            dense: true,
            title: Text(t.text, style: TextStyle(fontSize: 12)),
            subtitle: Text('${t.confidence.toStringAsFixed(2)} - ${t.timestamp}', style: TextStyle(fontSize:
10)),
          ),
        ),
      ],
    ),
  );
}
}

```

Generate the complete debug tools with mock data, testing utilities, and performance monitoring.

...

---

## DEPLOYMENT & DOCUMENTATION PROMPTS

### Prompt 14: Build Configuration & Deployment

...

Create build configuration and deployment scripts for the Fish Ledger app:



FILE: android/app/build.gradle

#### ANDROID BUILD CONFIGURATION:

```
android {
    compileSdkVersion 34

    defaultConfig {
        applicationId "com.fishledger.app"
        minSdkVersion 21
        targetSdkVersion 34
        versionCode 1
        versionName "1.0.0"

        // Enable multidex for Firebase
        multiDexEnabled true
    }

    buildTypes {
        release {
            minifyEnabled true
            shrinkResources true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'

            // Sign with release key
            signingConfig signingConfigs.release
        }

        debug {
            applicationIdSuffix ".debug"
            debuggable true
        }
    }

    // Configure flavors for different environments
    flavorDimensions "environment"
    productFlavors {
        dev {
            dimension "environment"
            applicationIdSuffix ".dev"
            versionNameSuffix "-dev"
        }

        prod {
            dimension "environment"
        }
    }
}

dependencies {
    // Firebase
    implementation platform('com.google.firebase:firebase-bom:32.7.0')
    implementation 'com.google.firebase:firebase-firestore'
    implementation 'com.google.firebase:firebase-auth'
    implementation 'com.google.firebase:firebase-analytics'
```

```
// Multidex
implementation 'androidx.multidex:multidex:2.0.1'
}
```

PROGUARD RULES (android/app/proguard-rules.pro):

```
# Firebase
-keepattributes Signature
-keepattributes *Annotation*
-keepattributes EnclosingMethod
-keepattributes InnerClasses

# Keep model classes
-keep class com.fishledger.app.models.** { *; }

# Flutter
-keep class io.flutter.app.** { *; }
-keep class io.flutter.plugin.** { *; }
-keep class io.flutter.util.** { *; }
-keep class io.flutter.view.** { *; }
-keep class io.flutter.** { *; }
-keep class io.flutter.plugins.** { *; }
```

ANDROID MANIFEST (android/app/src/main/AndroidManifest.xml):

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Permissions -->
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.RECORD_AUDIO"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>

    <application
        android:name="${applicationName}"
        android:label="Fish Ledger"
        android:icon="@mipmap/ic_launcher"
        android:requestLegacyExternalStorage="true">

        <!-- Main Activity -->
        <activity
            android:name=".MainActivity"
            android:launchMode="singleTop"
            android:theme="@style/LaunchTheme"

            android:configChanges="orientation|keyboardHidden|keyboard|screenSize|smallestScreenSize|locale|layoutDirection|fontScale|screenLayout|density|uiMode"
            android:hardwareAccelerated="true"
            android:windowSoftInputMode="adjustResize"
            android:exported="true">

            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
```

```

</activity>

<!-- Background Service -->
<service
    android:name=".AudioProcessingService"
    android:foregroundServiceType="microphone"
    android:exported="false"/>
</application>
</manifest>

```

BUILD SCRIPT (scripts/build.sh):

```

#!/bin/bash

echo "🔨 Building Fish Ledger App..."

# Clean previous builds
flutter clean
flutter pub get

# Run code generation
flutter pub run build_runner build --delete-conflicting-outputs

# Build APK
echo "📦 Building APK..."
flutter build apk --release --split-per-abi

# Build App Bundle
echo "📦 Building App Bundle..."
flutter build appbundle --release

echo "✅ Build complete!"
echo "APK location: build/app/outputs/flutter-apk/"
echo "Bundle location: build/app/outputs/bundle/"

```

DEPLOYMENT CHECKLIST:

Create a deployment checklist file: DEPLOYMENT.md

# Fish Ledger Deployment Checklist

## Pre-Deployment

- [ ] Update version in pubspec.yaml
- [ ] Update versionCode and versionName in build.gradle
- [ ] Test on multiple devices (min 3 different phones)
- [ ] Run accuracy tests (target: >60%)
- [ ] Test battery consumption (full day test)
- [ ] Verify Firebase configuration
- [ ] Check API keys are properly secured
- [ ] Review app permissions
- [ ] Test offline functionality
- [ ] Verify data sync works correctly

## Build

- [ ] Run `flutter clean`
- [ ] Run `flutter pub get`
- [ ] Build release APK: `flutter build apk --release`
- [ ] Build app bundle: `flutter build appbundle --release`
- [ ] Sign APK with release key
- [ ] Test release build on device

## ## Firebase Setup

- [ ] Create Firebase project
- [ ] Enable Firestore Database
- [ ] Set up Firestore security rules
- [ ] Enable Anonymous Authentication
- [ ] Add Android app to Firebase
- [ ] Download and add google-services.json
- [ ] Test Firebase connection

## ## Testing

- [ ] Install release build on test device
- [ ] Test full user flow:
  - [ ] Onboarding
  - [ ] Permission requests
  - [ ] Start/stop listening
  - [ ] Transaction detection
  - [ ] Manual entry
  - [ ] Transaction list
  - [ ] Settings changes
- [ ] Test with real fish seller (1+ days)
- [ ] Verify data persistence
- [ ] Test network errors/offline mode
- [ ] Battery consumption check

## ## Documentation

- [ ] Update README.md
- [ ] Create user manual (PDF)
- [ ] Create video tutorial
- [ ] Prepare demo video
- [ ] Document known issues

## ## Distribution

- [ ] Create Google Play Console account (if publishing)
- [ ] Prepare store listing (title, description, screenshots)
- [ ] Upload APK/Bundle to internal testing
- [ ] Share with beta testers
- [ ] Collect feedback
- [ ] Iterate based on feedback

Generate the complete build configuration, scripts, and deployment documentation.

...

---

### Prompt 15: Complete Documentation Package

...

Create comprehensive documentation for the Fish Ledger app:

FILE: README.md

# Fish Ledger - Voice-Activated Digital Bookkeeping for Fish Sellers

## Overview

Fish Ledger is a mobile application that automatically records fish market sales by listening to natural conversations between sellers and customers. Designed specifically for tabletop fish sellers in Ghana.

## Features

- 🎤 **Automatic Transaction Recording**: Listen to sales conversations and record transactions automatically
- 📊 **Daily Sales Summary**: View total sales, transaction count, and best-selling fish
- 📴 **Offline-First**: Works without internet connection
- 🌐 **Multi-Language**: Supports English, Twi, and code-switching
- 🔒 **Privacy-Focused**: All processing happens on-device
- ☁️ **Cloud Backup**: Optional Firebase sync for data safety

## Technology Stack

- **Framework**: Flutter 3.x
- **Backend**: Firebase (Firestore, Auth, Analytics)
- **Speech Recognition**: Google Cloud Speech-to-Text API
- **Audio Processing**: record, WebRTC VAD
- **State Management**: Provider
- **Local Storage**: SQLite, SharedPreferences

## Prerequisites

- Flutter SDK 3.10+
- Android Studio / Xcode
- Firebase account
- Google Cloud account (for Speech-to-Text API)

## Installation

### 1. Clone Repository

```
```bash
git clone https://github.com/yourusername/fish-ledger.git
cd fish-ledger
```
```

### 2. Install Dependencies

```
```bash
flutter pub get
```
```

### 3. Firebase Setup

1. Create a Firebase project at <https://console.firebase.google.com>
2. Add an Android app to your Firebase project

3. Download `google-services.json` and place it in `android/app/`
4. Enable Firestore Database and Anonymous Authentication

### ### 4. Google Cloud Speech API Setup

1. Go to <https://console.cloud.google.com>
2. Enable Speech-to-Text API
3. Create API credentials
4. Add API key to `lib/config/api\_keys.dart`:

```
```dart
class ApiKeys {
  static const String googleCloudSpeechApiKey = 'YOUR_API_KEY_HERE';
}
```
```

**\*\*IMPORTANT\*\*:** Never commit API keys to version control!

### ### 5. Run the App

```
```bash
flutter run
```
```

## ## Project Structure

```
...
fish_ledger/
├── lib/
│   ├── main.dart          # App entry point
│   ├── models/            # Data models
│   │   ├── transaction.dart
│   │   ├── user_profile.dart
│   │   └── audio_segment.dart
│   ├── services/          # Core services
│   │   ├── audio_service.dart
│   │   ├── speech_recognition_service.dart
│   │   ├── transaction_detection_service.dart
│   │   ├── firebase_service.dart
│   │   └── processing_pipeline.dart
│   ├── screens/           # UI screens
│   │   ├── home_screen.dart
│   │   ├── transaction_list_screen.dart
│   │   ├── settings_screen.dart
│   │   └── onboarding_screen.dart
│   ├── widgets/           # Reusable widgets
│   │   ├── transaction_card.dart
│   │   ├── daily_summary.dart
│   │   └── listening_indicator.dart
│   ├── utils/             # Utilities
│   │   ├── constants.dart
│   │   ├── pattern_matcher.dart
│   │   └── debug_tools.dart
│   ├── android/           # Android-specific code
│   ├── ios/               # iOS-specific code
│   └── test/              # Unit and widget tests
```

```
└── assets/          # Images, fonts, etc.
...
```

## ## How It Works

1. **Audio Capture**: App continuously listens to conversations through the phone's microphone
2. **Voice Activity Detection**: Identifies when speech is occurring
3. **Speech Recognition**: Converts audio to text using Google Cloud Speech-to-Text
4. **Pattern Matching**: Analyzes text for transaction patterns (price, fish type, confirmation)
5. **Transaction Detection**: Uses state machine to identify complete transactions
6. **Data Storage**: Saves transactions locally (SQLite) and syncs to Firebase

## ## Usage Guide

### ### For Sellers

1. **First Time Setup**
  - Open app and complete onboarding
  - Grant microphone permission
  - Enter business name and location
  - Select common fish types
2. **Daily Use**
  - Tap "Start Listening" at beginning of day
  - Place phone nearby (within 2 meters)
  - Conduct sales normally
  - App records transactions automatically
  - Tap "Stop Listening" at end of day
3. **View Sales**
  - See today's total on home screen
  - Tap "View All" for full transaction list
  - Review and edit transactions if needed
4. **Manual Entry**
  - Tap + button if transaction was missed
  - Enter fish type, price, and quantity
  - Tap Save

### ### For Developers

#### #### Running Tests

```
``bash
# Unit tests
flutter test

# Integration tests
flutter test integration_test/

# Accuracy tests
flutter run lib/utils/debug_tools.dart
...
```

#### #### Debug Mode

Enable debug overlay by:

1. Add `--debug` flag when running
2. Tap debug button (bottom right)
3. View transcripts, accuracy, and performance

#### #### Adding New Fish Types

Edit `lib/Utils/pattern\_matcher.dart`:

```
``dart
static const Map<String, List<String>> fishPatterns = {
  'new_fish': ['name1', 'name2', 'local_name'],
  // ...
};
...

```

#### #### Adding New Languages

1. Collect sample transactions in target language
2. Train speech recognition model
3. Add language patterns to `TransactionDetectionService`
4. Update UI translations

### ## Configuration

#### ### Settings (lib/config/settings.dart)

```
``dart
class AppConfig {
  // Audio settings
  static const int sampleRate = 16000;
  static const int bufferSize = 8192;

  // Detection settings
  static const double autoSaveThreshold = 0.8;
  static const Duration transactionTimeout = Duration(minutes: 2);

  // Performance settings
  static const int maxRecentTranscripts = 10;
  static const int maxTransactionsPerPage = 20;
}
...

```

### ## Testing with Real Users

See [TESTING.md](TESTING.md) for detailed testing protocols.

#### ### Quick Test Checklist

- [ ] App starts successfully
- [ ] Microphone permission granted
- [ ] Audio recording works
- [ ] Speech recognition returns text
- [ ] Sample transaction detected correctly
- [ ] Transaction saved to database
- [ ] UI updates with new transaction



- ☐ App survives full day of use
- ☐ Battery consumption acceptable (<20% per day)

## ## Known Issues

1. **Background noise**: High noise environments may affect accuracy
2. **Battery usage**: Continuous listening uses 10-15% battery per hour
3. **API costs**: Google Speech API has usage limits (60 min/month free)
4. **Multiple speakers**: Difficulty separating seller from customers
5. **Two numbers**: Limited recognition of Two number words

## ## Roadmap

- ☒ MVP with basic transaction detection
- ☒ English + Twi support
- ☐ Add Ga and Ewe languages
- ☐ Improved speaker identification
- ☐ Customer recognition (repeat buyers)
- ☐ Inventory tracking
- ☐ Financial insights and predictions
- ☐ Microfinance integration
- ☐ Multi-seller support

## ## Contributing

We welcome contributions! Please see [\[CONTRIBUTING.md\]](#)(CONTRIBUTING.md) for guidelines.

## ## License

[Your chosen license]

## ## Support

- Email: [support@fishledger.app](mailto:support@fishledger.app)
- WhatsApp: [Your number]
- Website: <https://fishledger.app>

## ## Acknowledgments

- Built with inspiration from Martin Luther's translation principles
- Designed for Ghana's informal fish market economy
- Special thanks to all test sellers who provided feedback

---

FILE: USER\_MANUAL.md

## # Fish Ledger User Manual

### ## Getting Started

#### ### Installation

1. Download Fish Ledger from Google Play Store
2. Open the app
3. Follow the setup wizard

### ### First Time Setup

#### #### Step 1: Welcome Screen

- Read about how the app works
- Tap "Get Started"

#### #### Step 2: Permissions

- The app needs microphone access to listen to your sales
- Tap "Allow" when prompted
- This is safe - only you can see your data

#### #### Step 3: Business Information

- Enter your business name (optional)
- Enter your market location (optional)
- Select the types of fish you sell most often
- Tap "Start Using App"

### ## Daily Use

#### ### Starting Your Day

1. Open Fish Ledger app
2. Tap the large green "START LISTENING" button
3. Place your phone nearby (on table, in pocket, etc.)
4. The app will show a small pulsing dot to indicate it's listening

#### ### During the Day

- Conduct your sales normally
- Talk to customers like you always do
- The app quietly records each sale
- You'll see a brief flash when a sale is detected
- Your total sales update automatically

#### ### Ending Your Day

1. Tap the large red "STOP LISTENING" button
2. Review your day's sales
3. The app will show:
  - Total money earned
  - Number of sales
  - Best-selling fish

### ## Viewing Your Sales

#### ### Today's Summary

On the home screen, you can see:

- Total sales amount (large number)
- Number of transactions
- Last 3 sales

#### ### Full Transaction List

1. Tap "View All" button

2. See all your sales organized by date

3. Each sale shows:

- Time of sale
- Fish type
- Amount
- How it was recorded (automatic or manual)

### ### Filtering Sales

- Tap "Today" to see today's sales
- Tap "Week" to see this week
- Tap "Month" to see this month
- Use search to find specific fish types

## ## Managing Transactions

### ### Editing a Sale

1. Find the transaction in your list
2. Tap on it
3. Tap "Edit"
4. Change the details
5. Tap "Save"

### ### Deleting a Sale

1. Find the transaction
2. Swipe left on it
3. Tap "Delete"
4. Confirm deletion


### ### Adding a Missed Sale

If the app didn't catch a sale:

1. Tap the + button (bottom right)
2. Select fish type
3. Enter amount
4. Enter quantity
5. Tap "Save"

## ## Settings

### ### Accessing Settings

- Tap the gear icon () at bottom of home screen

### ### Important Settings

#### **\*\*Auto Start Listening\*\***

- Turn ON to automatically start listening when you open the app
- Turn OFF if you want to manually control when the app listens

#### **\*\*Market Hours\*\***

- Set when your market day typically starts and ends
- App will only listen during these hours to save battery

## **\*\*Language\*\***

- Choose your primary language (English or Twi)
- App will better understand your conversations

## **\*\*Daily Summary\*\***

- Choose what time you want to receive your end-of-day summary

## **### Managing Your Data**

### **\*\*Export Data\*\***

- Tap "Export Data" to download your sales as a file
- You can open this in Excel or share it

### **\*\*Clear All Data\*\***

- ⚠ This permanently deletes everything!
- Only use if you want to start fresh
- Cannot be undone

## **## Tips for Best Results**

### **### Phone Placement**

- Keep phone within 2 meters of where you sell
- Avoid covering the microphone
- Don't put phone face-down

### **### Speaking Clearly**

- Speak at normal volume
- Don't whisper prices
- Try to reduce background music/radio

### **### Battery Life**

- Charge phone fully before market day
- Consider bringing a power bank
- The app uses about 10-15% battery per hour

### **### If App Misses Sales**

- Add them manually using the + button
- Check that microphone permission is still enabled
- Make sure app is still listening (look for pulsing dot)

## **## Troubleshooting**

### **### App Not Detecting Sales**

#### **\*\*Check:\*\***

- Is the app listening? (green button pressed, pulsing dot visible)
- Is microphone permission enabled?
- Is phone close enough to conversations?
- Is background noise too loud?

#### **\*\*Try:\*\***

- Restart the app
- Move phone closer
- Reduce background noise
- Add sales manually and report the issue

### ### App Crashing or Freezing

1. Force close the app
2. Clear app cache (in phone settings)
3. Restart your phone
4. Reinstall the app if problem persists

### ### Battery Draining Fast

- Make sure you tap "Stop Listening" when not selling
- Enable "Smart Listening" in settings
- Close other apps running in background
- Consider getting a power bank

### ### Can't See My Sales

- Check your internet connection (for cloud sync)
- Make sure you're looking at the correct date range
- Try pulling down to refresh the list

## ## Privacy & Security

### ### Your Data is Safe

- Only you can see your sales data
- No audio recordings are saved
- All data stays on your phone unless you enable cloud backup
- We never share your information with anyone

### ### Cloud Backup

- Optional feature in Settings
- Backs up your sales to the cloud
- Helps if you lose your phone or get a new one
- Still only accessible to you

## ## Getting Help

### ### In-App Help

- Tap "Help & Tutorial" in Settings
- Watch video tutorials
- Read FAQ

### ### Contact Support

- Email: support@fishledger.app
- WhatsApp: [Your number]
- Phone: [Your number]

### ### Report a Problem

1. Go to Settings
2. Tap "Report a Problem"
3. Describe what happened

4. Include screenshots if possible

## ## Frequently Asked Questions

**Q: Does this app cost money?**

A: [Your pricing model]

**Q: Will this use a lot of my data?**

A: Very little. Most processing happens on your phone. Only syncs small amounts of data.

**Q: What if I forget to start listening?**

A: Enable "Auto Start Listening" in Settings. You can also add missed sales manually.

**Q: Can other people see my sales?**

A: No. Your data is completely private and only visible to you.

**Q: What happens if my phone dies?**

A: If you enabled cloud backup, your data is safe. Otherwise, you'll lose transactions that weren't synced.

**Q: Can I use this for other products?**

A: Currently designed for fish, but you can add custom products in Settings.

**Q: How accurate is the app?**

A: Typically 60-70% of sales are detected automatically. You can always add missed sales manually.

**Q: Does the app work without internet?**

A: Yes! Most features work offline. You need internet only for cloud backup and updates.

---

Generate the complete documentation package including README, user manual, testing guides, and contribution guidelines.

...

---

## ## FINAL INTEGRATION PROMPT

### ### Prompt 16: Complete App Assembly

...

Using all the components created above, assemble the complete Fish Ledger application with proper integration:

#### INTEGRATION CHECKLIST:

##### 1. **Project Initialization**

- Create Flutter project structure
- Add all dependencies to pubspec.yaml
- Configure Firebase (google-services.json)
- Set up Android permissions and manifest

##### 2. **Core Models**

- Implement Transaction model with Firestore serialization
- Implement UserProfile model

- Implement AudioSegment model
- Add proper validation and null safety

### 3. **Services Layer**

- AudioService: Continuous recording with VAD
- SpeechRecognitionService: Google Cloud Speech-to-Text integration
- TransactionDetectionService: Pattern matching and state machine
- FirebaseService: All CRUD operations
- ProcessingPipeline: Orchestrate all services

### 4. **UI Layer**

- HomeScreen: Main dashboard with start/stop controls
- TransactionListScreen: Filterable transaction history
- SettingsScreen: All configuration options
- OnboardingScreen: First-time user experience

### 5. **Integration Points**

- Wire services through Provider
- Connect UI to services
- Implement state management
- Add error handling throughout

### 6. **Testing & Debugging**

- Add debug overlay
- Implement accuracy testing
- Add performance monitoring
- Create mock data generators

### 7. **Build & Deploy**

- Configure release build
- Set up ProGuard rules
- Create deployment scripts
- Write documentation

### CRITICAL CONNECTIONS:

```

dart
// main.dart integration example
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();

  // Initialize all services
  final services = await ServiceInitializer.initialize();

  runApp(
    MultiProvider(
      providers: [
        Provider<AudioService>.value(value: services.audioService),
        Provider<SpeechRecognitionService>.value(value: services.speechService),
        Provider<TransactionDetectionService>.value(value: services.detectionService),
        Provider<FirebaseService>.value(value: services.firebaseService),
        Provider<ProcessingPipeline>.value(value: services.pipeline),
        ChangeNotifierProvider<AppState>(create: (_) => AppState()),
      ],
      child: FishLedgerApp(),
    ),
  );
}

```

```

    },
);
}

// HomeScreen integration example
class _HomeScreenState extends State<HomeScreen> {
  late ProcessingPipeline pipeline;
  late AppState appState;

  @override
  void initState() {
    super.initState();
    pipeline = context.read<ProcessingPipeline>();
    appState = context.read<AppState>();

    // Listen to pipeline for new transactions
    _setupListeners();
  }

  void _setupListeners() {
    // Connect pipeline events to app state
    pipeline.onTransactionDetected.listen((transaction) {
      appState.addTransaction(transaction);
    });
  }

  Future<void> _toggleListening() async {
    if (pipeline.getStatus().isRunning) {
      await pipeline.stop();
      appState.stopListening();
    } else {
      // Check permissions first
      final hasPermission = await PermissionManager.requestMicrophonePermission();
      if (hasPermission) {
        await pipeline.start();
        appState.startListening();
      } else {
        _showPermissionDeniedDialog();
      }
    }
    setState(() {});
  }
}
...

```

#### STARTUP SEQUENCE:

1. Firebase initialization
2. Anonymous authentication
3. Service creation and initialization
4. Check onboarding status
5. Request permissions
6. Navigate to appropriate screen
7. Start background services if auto-start enabled

#### ERROR HANDLING STRATEGY:



```

```dart
class ErrorHandler {
  static void handleError(dynamic error, {required String context}) {
    print('Error in $context: $error');

    // Categorize error
    if (error is FirebaseException) {
      _handleFirebaseError(error);
    } else if (error is PlatformException) {
      _handlePlatformError(error);
    } else {
      _handleGenericError(error);
    }

    // Log to analytics (if enabled)
    // Don't crash the app
  }
}
```

```

#### PERFORMANCE OPTIMIZATION:

```

```dart
class PerformanceOptimizer {
  // Lazy load heavy services
  static Future<void> preloadServices() async {
    // Pre-initialize speech recognition
    // Load pattern matching models
    // Cache common fish types
  }

  // Battery optimization
  static void optimizeBatteryUsage() {
    // Reduce processing when phone is stationary
    // Lower sampling rate during quiet periods
    // Stop completely during phone sleep
  }
}
```

```

#### FINAL TESTING:

Before considering the app complete, verify:

- [ ] App starts without errors
- [ ] Onboarding flow works
- [ ] Permissions requested properly
- [ ] Audio recording functions
- [ ] Speech recognition returns text
- [ ] Transactions detected with >60% accuracy
- [ ] Firebase sync works
- [ ] UI updates in real-time
- [ ] App survives full day use
- [ ] Battery usage <20% per day
- [ ] All screens navigable

- [ ] Settings persist
- [ ] Manual entry works
- [ ] Export data works
- [ ] App handles errors gracefully

Generate the complete, integrated Fish Ledger application with all components properly connected and working together.

...

---

## ## USAGE INSTRUCTIONS FOR KIRO

To use these prompts with Kiro effectively over your 27-day development period:

### ### RECOMMENDED WORKFLOW

#### #### Week 1: Foundation (Days 1-7)

##### **\*\*Day 1: Project Setup\*\***

...

Give Kiro: Prompt 1 (Project Setup and Structure)

Expected output: Complete Flutter project with Firebase configured

Validation: Run `flutter run` successfully

...

##### **\*\*Day 2: Data Models\*\***

...

Give Kiro: Prompt 2 (Data Models)

Expected output: All model classes with serialization

Validation: Models can serialize/deserialize from JSON

...

##### **\*\*Day 3-4: Audio Service\*\***

...

Give Kiro: Prompt 3 (Audio Recording Service)

Expected output: Working AudioService class

Validation: Can record audio continuously for 2 hours

Test: Record in noisy environment, check battery usage

...

##### **\*\*Day 5-6: Speech Recognition\*\***

...

Give Kiro: Prompt 4 (Speech Recognition Service)

Expected output: SpeechRecognitionService with Google Cloud integration

Validation: Transcribes your 20 test recordings with >70% accuracy

Test: Speak sample phrases, verify transcript correctness

...

##### **\*\*Day 7: Transaction Detection\*\***

...

Give Kiro: Prompt 5 (Transaction Detection Service)

Expected output: TransactionDetectionService with pattern matching

Validation: Detects 12+ out of 20 test transactions

Test: Feed sample transcripts, verify extraction of price/fish type

...

## #### Week 2: Integration (Days 8-14)

### \*\*Day 8: Firebase Service\*\*

...

Give Kiro: Prompt 6 (Firebase Service)

Expected output: Complete FirebaseService with CRUD operations

Validation: Can save and retrieve transactions

Test: Save 10 transactions, verify they appear in Firestore console

...

### \*\*Day 9-10: UI Screens\*\*

...

Give Kiro: Prompts 7, 8, 9 (Home, Transaction List, Settings)

Expected output: Three main screens with basic functionality

Validation: Can navigate between screens

Test: UI is readable in sunlight, buttons are large enough

...

### \*\*Day 11: Onboarding\*\*

...

Give Kiro: Prompt 10 (Onboarding Screen)

Expected output: 4-screen onboarding flow

Validation: Complete onboarding saves user profile

Test: Go through onboarding, verify data saved

...

### \*\*Day 12-13: Main Integration\*\*

...

Give Kiro: Prompt 11 (Main App Integration)

Expected output: main.dart with all services connected

Validation: App starts and reaches home screen

Test: All screens accessible, no crashes on navigation

...

### \*\*Day 14: Processing Pipeline\*\*

...

Give Kiro: Prompt 12 (Core Processing Loop)

Expected output: ProcessingPipeline connecting all services

Validation: Audio → Transcript → Detection → Storage works end-to-end

Test: Simulate transaction conversation, verify it saves to database

...

## #### Week 3: Testing & Polish (Days 15-21)

### \*\*Day 15: Debug Tools\*\*

...

Give Kiro: Prompt 13 (Testing & Debugging Tools)

Expected output: Debug overlay, accuracy tester, performance monitor

Validation: Can run accuracy tests and see results

Test: Run test scenarios, measure accuracy

...

### \*\*Day 16-18: Real-World Testing\*\*

- Deploy to test device

- Visit fish market

- Test with real seller for 3 days
- Collect failure data
- Iterate with bug fixes

#### **\*\*Day 19-21: Improvements\*\***

- Give Kiro specific bug fix prompts based on real-world findings
- Example: "Fix issue where Twi numbers aren't recognized"
- Example: "Improve noise filtering for very crowded markets"

#### **#### Week 4: Finalization (Days 22-27)**

##### **\*\*Day 22-23: Build Configuration\*\***

...

Give Kiro: Prompt 14 (Build Configuration & Deployment)

Expected output: Release build configuration, deployment scripts

Validation: Can build signed release APK

Test: Install release APK on fresh device

...

##### **\*\*Day 24-25: Documentation\*\***

...

Give Kiro: Prompt 15 (Complete Documentation Package)

Expected output: README, User Manual, Testing guides

Validation: Documentation is clear and complete

Test: Have non-technical person read user manual

...

##### **\*\*Day 26: Final Assembly\*\***

...

Give Kiro: Prompt 16 (Complete App Assembly)

Expected output: Fully integrated, polished application

Validation: Pass all items in final testing checklist

Test: Full day real-world test with seller

...

##### **\*\*Day 27: Presentation\*\***

- Create demo video
- Prepare presentation
- Document lessons learned
- Plan next steps

---

#### **## HOW TO GIVE PROMPTS TO KIRO**

##### **### Format for Each Prompt:**

...

I'm building a Fish Ledger app for Ghanaian fish sellers.

Context: [Brief description of what you've built so far]

Task: [Copy the specific prompt from above]

Requirements:

- Use Flutter 3.x

- Integrate with Firebase
- Follow best practices for null safety
- Add proper error handling
- Include comments explaining key logic

Please generate the complete, production-ready code.

...

### Example for Day 3:

...

I'm building a Fish Ledger app for Ghanaian fish sellers.

Context: I have the project structure set up and data models created. Now I need the audio recording functionality.

Task: Create a comprehensive audio recording service (lib/services/audio\_service.dart) with the following requirements:

[Copy Prompt 3 content here]

Requirements:

- Use Flutter 3.x and the 'record' package
- Implement proper permission handling
- Add Voice Activity Detection
- Optimize for battery life
- Include error handling for all edge cases
- Add detailed comments

Please generate the complete, production-ready AudioService class.

...

---

## TIPS FOR WORKING WITH KIRO

### 1. **Be Specific About Context**

Always tell Kiro what you've already built:

- "I have the AudioService working"
- "I've tested speech recognition with 70% accuracy"
- "The UI screens are complete but not connected yet"

### 2. **Request Iterations**

If the first output isn't perfect:

...

The AudioService you created has an issue with [specific problem].

Please fix [specific issue] while maintaining [what works].

...

### 3. **Ask for Explanations**

...

Can you explain how the TransactionStateMachine works?

How does the confidence scoring algorithm determine if a transaction is valid?

...

### 4. **Request Testing Code**

Generate unit tests for the TransactionDetectionService.  
Create integration tests for the complete audio → detection → storage flow.  
...

### 5. \*\*Get Debugging Help\*\*  
...

I'm getting this error: [paste error]  
In this context: [explain what you were doing]  
Here's the relevant code: [paste code]  
How do I fix this?  
...

### 6. \*\*Request Optimizations\*\*  
...

The app is using too much battery. Can you optimize the AudioService  
to reduce battery consumption while maintaining accuracy?  
...

## ## COMMON ISSUES & SOLUTIONS

### Issue: Firebase Not Connecting  
...

Kiro, I'm getting "Firebase not initialized" error.  
I've added google-services.json to android/app/.  
The error happens when: [describe when]  
Please help me debug the Firebase initialization.  
...

### Issue: Speech Recognition Accuracy Too Low  
...

Kiro, the speech recognition is only 40% accurate on my test recordings.  
The recordings have: [describe audio conditions]  
Please improve the SpeechRecognitionService to handle:  
- Background market noise  
- Ghanaian English accent  
- Code-switching between English and Twi  
...

### Issue: App Crashes After Running for 1 Hour  
...

Kiro, the app crashes after about 1 hour of continuous listening.  
The error log shows: [paste error]  
Please add memory management and cleanup to prevent memory leaks.  
...

### Issue: Transactions Not Saving to Firebase  
...

Kiro, transactions are detected but not saving to Firestore.  
The console shows: [paste console output]  
The transaction data looks like: [paste transaction]  
Please fix the FirebaseService save operation.  
...

## ## VALIDATION CHECKLIST

After each Kiro output, validate:

### ### Code Quality Checks

- [ ] Code compiles without errors
- [ ] No null safety warnings
- [ ] Proper error handling exists
- [ ] Comments explain complex logic
- [ ] Follows Dart/Flutter conventions

### ### Functionality Checks

- [ ] Feature works as specified
- [ ] Edge cases handled
- [ ] Performance acceptable
- [ ] Memory usage reasonable
- [ ] Battery impact minimal

### ### Integration Checks

- [ ] Works with existing code
- [ ] No breaking changes
- [ ] Services connect properly
- [ ] State management works
- [ ] UI updates correctly

## ## EMERGENCY SHORTCUTS

If running behind schedule, use these streamlined approaches:

### ### Day 10 Checkpoint: Not Enough Working

**\*\*Quick Fix Strategy:\*\***

1. Focus on ONE core flow: Home screen → Start listening → Detect ONE transaction type (tilapia only) → Save → Display
2. Skip: Settings screen, transaction editing, multiple fish types, Twi language
3. Use hardcoded patterns instead of complex state machine
4. Prompt Kiro: "Create the simplest possible version that demonstrates the core concept"

### ### Day 17 Checkpoint: Real Testing Failing

**\*\*Quick Fix Strategy:\*\***

1. If accuracy < 40%: Add manual entry as primary method, voice as "bonus"
2. If battery issues: Reduce listening to button press mode (not continuous)
3. If crashes: Add try-catch around everything, log errors, continue anyway
4. Prompt Kiro: "Make this more robust - prioritize stability over features"

### ### Day 24 Checkpoint: Not Ready for Demo

**\*\*Quick Fix Strategy:\*\***

1. Create pre-recorded demo video (not live demo)
2. Use best-case scenarios
3. Have manual backup for every feature
4. Prompt Kiro: "Generate a demo script and mock data that showcases the vision"

## ## FINAL 48-HOUR SPRINT (Days 26-27)

If you reach Day 26 and need to pull everything together:

### ### Hour-by-Hour Plan

#### \*\*Day 26:\*\*

- Hours 1-4: Final bug fixes (give Kiro specific issues)
- Hours 5-8: Create demo video with working transactions
- Hours 9-12: Write presentation and documentation
- Hours 13-16: Practice demo, identify weak points

#### \*\*Day 27:\*\*

- Hours 1-4: Polish presentation
- Hours 5-8: Create backup demos for failure scenarios
- Hours 9-12: Final testing and validation
- Hours 13-16: Presentation delivery

---

## ## SUCCESS METRICS

By Day 27, you should have:

### ### Minimum Success (60% complete)

- ☐ App runs without crashing
- ☐ Can record audio
- ☐ Can detect SOME transactions (even if only 40% accurate)
- ☐ Can save transactions to Firebase
- ☐ Basic UI shows sales total
- ☐ Demo video shows concept working

### ### Good Success (80% complete)

- ☐ 60%+ detection accuracy
- ☐ Complete UI (all screens functional)
- ☐ Works for full day without crashing
- ☐ Battery usage acceptable (<20% per day)
- ☐ One seller willing to continue using
- ☐ Clear documentation

### ### Great Success (100% complete)

- ☐ 70%+ detection accuracy
- ☐ Polished UI and UX
- ☐ Multiple sellers interested
- ☐ Both English and Twi working
- ☐ Partnership interest from microfinance/NGO
- ☐ Clear path to scaling

---

## ## KIRO BEST PRACTICES

### ### DO:

- ✔ Give complete context with each prompt
- ✔ Validate output before moving to next step



- ✅ Request explanations for complex code
- ✅ Ask for tests alongside implementation
- ✅ Iterate on outputs that aren't quite right
- ✅ Save all Kiro outputs for reference

### DON'T:

- ❌ Give all 16 prompts at once
- ❌ Skip validation steps
- ❌ Assume code works without testing
- ❌ Move forward with broken components
- ❌ Forget to integrate components
- ❌ Wait until Day 26 to test

---

## ## POST-27-DAY ROADMAP

If you successfully complete the 27-day sprint, next steps:

### ### Immediate (Days 28-35)

- Continue testing with 1-3 sellers
- Fix critical bugs discovered
- Improve accuracy based on real data
- Add most-requested features

### ### Short-term (Months 2-3)

- Expand to 10-20 sellers
- Add Ga and Ewe languages
- Implement advanced features (inventory, analytics)
- Build partnerships

### ### Long-term (Months 4-12)

- Scale to 100+ sellers
- Add microfinance integration
- Expand to other product types
- Launch in multiple cities

---

## ## CONCLUSION

These 16 prompts provide a complete blueprint for building the Fish Ledger app in 27 days with Kiro's assistance. The key to success is:

1. **\*\*Follow the sequence\*\*** - Each prompt builds on previous ones
2. **\*\*Validate thoroughly\*\*** - Test each component before moving on
3. **\*\*Iterate quickly\*\*** - Don't get stuck on perfection
4. **\*\*Test with real users\*\*** - By Day 15, get to market
5. **\*\*Focus on proof-of-concept\*\*** - 60% accuracy that works is better than 100% accuracy that doesn't exist

Remember Luther's wisdom: He didn't wait for perfection before testing his translation with real people. Get to market early, learn from real users, and iterate based on authentic feedback.

**\*\*Your mantra for the next 27 days:\*\***

"Working demo beats perfect plan. Real feedback beats assumptions. Done is better than perfect."

Good luck building Fish Ledger! 🐟📱🚀