Feature Replication Guide

This document contains analysis and "prompts" to replicate specific features of the Janani AI system. Use these prompts to instruct an AI (or developer) to rebuild the feature from scratch.

---

1. Custom AI API Configuration (Smart Proxy Support)

Context & Problem

The application needed to support both: 1. **Standard Google Gemini Keys** (Official API). 2. **Custom Proxy Keys** (e.g., OneBrain app) which require a custom Base URL. 3. **Deployment Variables**: Render/Docker environments where variables might be missing or mismatched.

The original code was hardcoded to the Proxy URL, causing immediate failure when a user tried to use a standard Google Key.

Analysis

- **Goal**: Flexible configuration that "just works" for both standard and proxy users.

  Mechanism:

  1. Load `GEMINI_BASE_URL` from environment (for manual overrides).

  2. Implement "Smart Detection": Check if key starts with `AIza` (Google Standard).

  3. If Standard Key -> Force Official Google URL (Override env/default).

4. If Non-Standard -> Use Env URL or Default Proxy.

Replication Prompt

> "I need to configure a FastAPI service (`ai_service.py`) to connect to Google Gemini.

> **Requirements:** 1. It must support a default **Proxy URL** (e.g., `onebrain.app`) for internal users. 2. It must ALSO support standard **Google API Keys** (`AIza...`) provided by external users. 3. **Smart Switching**: - If the configured API Key starts with `AIza`, ignore the proxy URL and force the client to use the official `generativelanguage.googleapis.com` endpoint. - Otherwise, use the configured `GEMINI_BASE_URL` or default proxy. 4. **Environment Variables**: Ensure `GEMINI_BASE_URL` is exposed in `config.py` so it can be manually set in production (Render) if needed for a *different* custom proxy."

---

2. Enhanced Error Reporting (Live Diagnostics)

Context & Problem

When the AI service failed on the live server, the chatbot returned a generic "AI not configured" message. This hid the actual error (e.g., `401 Unauthorized`, `ConnectTimeout`), making remote debugging impossible without server logs.

Analysis

- **Goal**: Expose specific technical errors to the user (or admin) in the chat interface when critical failures occur.

  Mechanism:

  1. Catch exceptions in the fallback logic.

2. Store the `last_error` message.

3. Append the error details (e.g., "Technical Error: {details}") to the final user-facing failure message.

Replication Prompt

> "Modifiy the `get_response` method in `AIService` to improve error visibility.

> **Requirements:** 1. When an API call fails (Gemini or DeepSeek), capture the specific Python exception message (e.g., `str(e)`). 2. If *all* fallback attempts fail, do NOT just return a generic 'Sorry' message. 3. Append the captured error details to the end of the response: `\n\nTechnical Error: <actual_error_message>`. 4. This allows me to diagnose deployment issues (like Auth failures) directly from the client UI."

---

3. Digital Midwife Core (Persona & Care Plans)

Context

The core of Janani is relevant, medically accurate advice delivered in a "Village Sister" (Apu) persona. It generates weekly care plans and assesses risk based on user profile data.

Analysis

- **Core Logic**: `MaternalRiskProfile` model tracks weeks, vitals, and conditions.

  Algorithms:

  - **Risk Calculation**: Deterministic logic (Age < 18, BMI > 30, BP > 140/90) -> `RiskLevel`.

- **Care Plan**: Week-by-week guide based on WHO ANC guidelines.

- **Persona**: AI Instructions to speak in "Cholitobhasha" (Standard Colloquial) Bengali, avoiding "Sadhu" (Formal) language.

Replication Prompt

> "Build a FastAPI router `midwife_router.py` to manage maternal health profiles and care plans.

> **Requirements:** 1. **Data Model**: Create a `MaternalRiskProfile` Pydantic model with fields for LMP (Last Menstrual Period), Age, BMI, Blood Pressure, and Hemoglobin. 2. **Risk Logic**: Implement a deterministic function that calculates `RiskLevel` (Low, Moderate, High, Critical). E.g., If Age < 18 OR BMI > 30, Risk = Moderate. If BP >= 140/90, Risk = High. 3. **Weekly Care Plan**: Create a service that generates valid WHO-based advice for a specific pregnancy week (1-42). 4. **Persona Injection**: Implement a `humanize_care_plan` endpoint that takes the dry clinical plan and uses an LLM (Gemini) to rewrite it as 'Janani Apu' - a warm, caring sister speaking colloquial Bengali."

---

4. Emergency Bridge & AR Assistant

Context

A critical feature for rural areas. It bridges the gap between symptom detection and hospital arrival, providing real-time AR guidance for offline support.

Analysis

Emergency Bridge:

- **Trigger**: Voice detection of keywords ("Bleeding", "Pain") or Triage result.

- **Response**: Immediate "Action Plan" (json) + Hospital Locator using `haversine` distance.

AR Guidance:

- **Data Structure**: Returns JSON configs for MediaPipe (Overlay info, target angles for body positions).

- **Offline Support**: Rules are served via `offline_rules.json` for when internet fails.

Replication Prompt

> "Create an `EmergencyBridgeService` that handles critical maternal emergencies.

> **Requirements:** 1. **Protocol Database**: Hardcode emergency protocols for Hemorrhage, Eclampsia, and Labor. Each must have `immediate_steps` (Bengali) and `do_not` actions. 2. **Location Service**: Implement a function to find the nearest hospital from a JSON list (`hospitals.json`) using the User's Latitude/Longitude. 3. **AR Logic**: Create an endpoint `/emergency/guidance/{type}` that returns JSON configuration for a frontend MediaPipe AR overlay. - Example Data: `{ type: 'position', target_angle: 30, instructions: ['Lie flat', 'Raise legs'] }`. 4. **Voice Guidance**: Generate a script for the AI to speak calmly, guiding the user through the first 3 steps while the ambulance is called."

---

5. Food Intelligence (Vision & RAG)

Context

Analyzes food images or text to determine safety during pregnancy, specifically checking for culturally relevant restrictions (e.g., Raw Papaya).

Analysis

Pipeline:

1. **Vision**: Analyze image content (Ingredients, Nutrition).
2. **RAG (Retrieval Augmented Generation)**: Look up the detected food in a vector/rules database for specific pregnancy warnings.
3. **Visual Menu**: Generate a meal plan with *images* (using Pollinations.ai or similar) to make it visually appealing.

Replication Prompt

> "Build a `FoodAnalysisRouter` with two main capabilities:

> **1. Safety Analysis (RAG): -** Input: Food Name or Image. **-** Process: Retrieve pregnancy safety data. Check for specific Bengali cultural restrictions (e.g., Pineapple, Raw Papaya). **-** Output: Safety Verdict (Safe/Unsafe/Caution) + Bengali Explanation.

> **2. Visual Menu Generator: -** Input: User Budget (BDT) + Trimester. **-** Process: Use an LLM to generate a 5-item daily menu. **- Critical**: For each item, generate a valid Image URL (e.g., using Pollinations.ai with a specific prompt) so the user sees a photo of the food, not just text."

---

6. Voice & Speech System (Polyglot)

Context

Essential for illiterate users. The system must speak Bengali fluently and understand mixed "Banglish" or dialect speech.

Analysis

- **Input (STT)**: Uses `SpeechRecognition` library (Google wrapper) for generic Bengali support.

  Output (TTS):

  - **Tier 1**: ElevenLabs (High quality, expensive). Use if Key exists.
  - **Tier 2**: gTTS (Google TTS, Free, Robotic). Fallback if Tier 1 fails/quota empty.

Replication Prompt

> "Implement a robust `SpeechService` logic for a Bengali voice bot.

> **Requirements:** 1. **Dual-Engine TTS**: - Primary: Check for an `ELEVENLABS_API_KEY`. If present, use ElevenLabs API for high-quality voice. - Fallback: Wrap the call in a try/except. If it fails (Auth/Quota), immediately fall back to `gTTS` (Google Text-to-Speech) so the app never stays silent. 2. **Audio Conversion**: Use `ffmpeg` (via `pydub`) to convert any incoming user audio (blob/webm) to WAV before processing, as browsers send various formats. 3. **Response**: The API should return an audio file stream (bytes) directly to the frontend."