# Telemedicine Platform — 30‑Hour Build Guide (Buildathon)

**Goal:** Build a working telemedicine MVP that supports regional languages, symptom assessment, basic drug-safety guidance, and clinician escalation — all within 30 hours using free tools where possible.

## Executive summary

A compact, tactical plan to deliver an end-to-end telemedicine prototype for a hackathon: voice/text in regional languages → transcription → GPT-4 based structured triage (function-calling) → safe drug warnings → clinician escalation path → health-education snippets. The approach emphasizes safety, cost-control (use of free tiers/fallbacks), and a show-ready demo.

## Problem statement

Many users in regional-language areas lack quick, accessible, and trustworthy primary-care guidance. Existing telemedicine solutions are often English-centric, costly, or not optimized for concise triage and health education in local languages. This project aims to deliver an accessible triage and telemedicine assistance prototype to bridge that gap.

## Solution (how it works)

1. **User Input:** Accept user queries as text or short voice clips in a regional language.
2. **Transcription:** Convert voice → text using Whisper (OpenAI or local alternative).
3. **Normalization & Translation (if needed):** Optionally translate to English for reasoning, or operate directly in the regional language.
4. **Symptom Assessment:** Call GPT-4 with a conservative system prompt and function-calling schema to produce structured triage output: triage\_level, probable\_conditions, red\_flags, drug\_warnings, recommended\_next\_steps.
5. **Safety Checks:** Run moderation and red-flag detection; force clinician escalation for high-risk cases.
6. **Doctor Dashboard:** A simple queue + chat/placeholder video connection for clinicians to take sessions.
7. **Health Education:** Generate short, local-language health-education snippets (2–3 bullets).
8. **Logging & Audit:** Store session metadata for demo playback and evaluation.

## 

## How will you use OpenAI APIs?

**Primary tasks:**

* **GPT-4 (chat completion, function-calling):** Produce concise, structured triage assessments and health-education text. Use few-shot examples and a strict system prompt to avoid hallucinations.
* **Whisper (ASR):** Transcribe short voice clips in regional languages. Optionally fallback to local open-source solutions for offline demos.
* **Moderation endpoint (optional):** Filter content for self-harm, violent content, or disallowed material before processing.

**Cost-control measures for hackathon:**

* Keep audio durations short (10–30s).
* Limit tokens returned by GPT-4 (concise outputs).
* Cache responses for repeated scenarios and offline playback.
* Use local open-source fallback models where applicable.

## Tech stack (free-focused)

* **Frontend:** React or Next.js (deploy to Vercel Hobby).
* **Backend:** FastAPI (Python). Deploy to a free hosting tier or use local ngrok for live demo.
* **DB:** Supabase (free tier) or local SQLite for the MVP.
* **Speech:** OpenAI Whisper (or whisper.cpp/local ASR fallback).
* **TTS/Translation:** Browser SpeechSynthesis or open-source Hugging Face models for demo.

## 30‑Hour Sprint Plan (hour-by-hour)

*Assumes a single developer; adjust if partnering.*

**Hour 0 – 1 (Setup)**

* GitHub repo and README.
* Vercel account for frontend, Supabase trial or local SQLite.

**Hour 1 – 3 (Scaffold UI)**

* Minimal React page: text input, Record button, language selector, results panel, Consult Doctor CTA.

**Hour 3 – 5 (Backend scaffold)**

* FastAPI endpoints: /api/transcribe, /api/assess, /api/doctor/queue. Add CORS and env-based keys.

**Hour 5 – 8 (STT pipeline)**

* Audio upload and POST to backend.
* Integrate Whisper API (if key present) or implement a mocked/local transcription path for demo.

**Hour 8 – 11 (GPT-4 symptom assessment)**

* Construct a conservative system prompt + 3 few-shot examples in the target language (or bilingual).
* Implement function-calling schema to enforce structured JSON responses.

**Hour 11 – 13 (UI rendering & polish)**

* Render triage badge, red flags, suggested next steps; display short explanation and disclaimers.

**Hour 13 – 15 (Doctor flow & queue)**

* Doctor dashboard: pending sessions list and a take session button that opens a chat view (mocked doctor account OK).

**Hour 15 – 17 (Drug interaction & dosage safety)**

* Add a conservative CSV-based drug-interaction lookup for common OTC medications; avoid exact dosing unless obvious and safe.

**Hour 17 – 18 (Translation & TTS fallback)**

* Provide optional translation for reasoning; use browser TTS or a local TTS model for playback.

**Hour 18 – 20 (Safety & moderation)**

* Add moderation checks and emergency escalation rules. Auto-flag self-harm or severe symptom messages.

**Hour 20 – 22 (Caching, logging & cost control)**

* Cache repeated queries; log interactions to DB for demo playback.

**Hour 22 – 24 (Test flows)**

* Validate 8–10 canonical scenarios (cold, possible fracture, chest pain, drug-interaction query, etc.).

**Hour 24 – 27 (Deploy & demo pages)**

* Deploy frontend to Vercel. Backend to a free host or run locally with ngrok. Add README run instructions.

**Hour 27 – 29 (Demo recording)**

* Record a 3–4 minute demo: regional-language input → transcription → triage → escalate to doctor → education snippet.

**Hour 29 – 30 (Final presentation prep)**

* Prepare 3 slides: Problem, Live Demo, Why it Wins. Add speaker notes.

## Minimal MVP features to impress judges

1. Real-time short voice input → accurate transcription and structured triage.
2. Obvious red-flag detection + Connect to Doctor flow.
3. Health-education snippet in the user language.
4. Simple doctor dashboard showing queue + session takeover.
5. Safety & cost controls (moderation, caching, offline fallback).

## Implementation tips & minimal code snippets

**FastAPI example (pseudo):**

from fastapi import FastAPI, UploadFile  
app = FastAPI()  
  
@app.post("/api/transcribe")  
async def transcribe(audio: UploadFile):  
 text = call\_whisper\_api\_or\_local(audio)  
 return {"text": text}  
  
@app.post("/api/assess")  
async def assess(payload: dict):  
 response = call\_openai\_chat\_completion(messages, functions=..., model="gpt-4")  
 return response

**Function-calling schema (concept):**

{  
 "name": "triage\_assessment",  
 "parameters": {  
 "type": "object",  
 "properties": {  
 "triage\_level": {"type":"string", "enum":["self-care","primary-care","urgent","emergency"]},  
 "probable\_conditions": {"type":"array","items":{"type":"string"}},  
 "red\_flags": {"type":"array","items":{"type":"string"}},  
 "drug\_warnings": {"type":"array","items":{"type":"string"}},  
 "explain\_short": {"type":"string"}  
 },  
 "required": ["triage\_level","explain\_short"]  
 }  
}

**Prompting pattern:**

* **System prompt:** “You are a conservative medical triage assistant. Never give definitive diagnosis. Always output JSON via function-calling and escalate when unsure or if red flags appear.”
* Include 3 short few-shot examples covering self-care, primary-care and emergency.

## Drug interactions and dosing guidance (safety-first)

* Use a curated CSV of common OTC drugs and serious interactions.
* Avoid precise dosing instructions unless they are widely accepted simple adult OTC doses (e.g., paracetamol 500–1000 mg every 4–6 h, not exceeding daily maximum — still add a disclaimer).
* Always advise: “Confirm dosage with packaging or a clinician.”

## Risk mitigation (legal & safety)

* Prominently show: “This is a prototype for triage/education only — not a replacement for clinical judgment.”
* Force escalation for red flags (chest pain, loss of consciousness, severe bleeding, severe breathing difficulty, self-harm intent).
* Limit PHI storage and avoid storing personal identifiers in demo logs.

## Feasibility & scaling plan

**30‑hour demo:** Achieve an end-to-end deterministic flow with caching and local fallbacks to minimize live API calls.

**Post-hackathon scale:** Add formal authentication, encrypted storage, audit trails, clinician onboarding, and paid OpenAI usage monitoring. Implement a human-in-the-loop review and certification with healthcare partners.

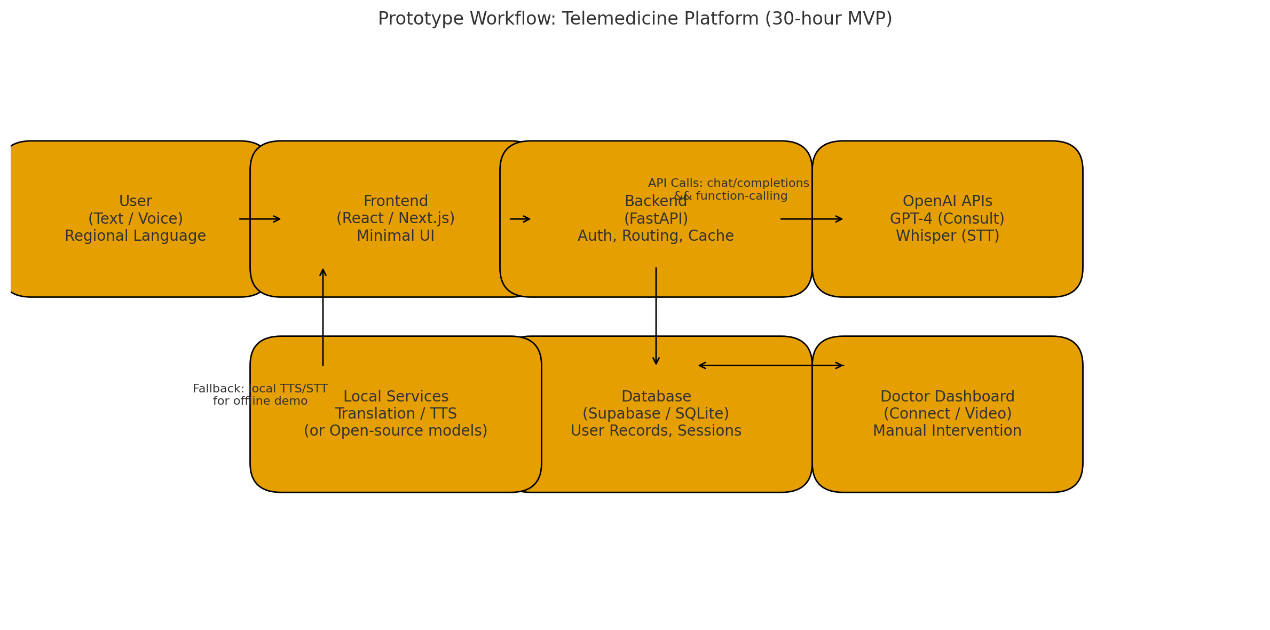
## Demo script (2–3 minutes)

1. 15s — Elevator pitch: problem & solution.
2. 60s — Live demo: speak in a regional language; show transcription, structured triage output (badge + red flags + suggested next steps).
3. 30s — Click Escalate → show doctor dashboard picking up the session.
4. 20s — Show autogenerated health-education snippet in the local language.
5. 10s — Close with safety & cost-control notes.

## Workflow & Prototype Images

Below are the workflow and prototype mockup images you can use in slides or the README.

*Workflow: User → Frontend → Backend → OpenAI APIs + DB + Doctor Dashboard + local fallbacks.*



*Prototype: Minimal UI wireframe showing text/voice input, language selector, results panel, and doctor CTA.*

## Next steps I can run now (tell me what you want)

* Generate final system prompt + 3 few-shot examples in English and a regional-language template (e.g., Telugu or Hindi).
* Scaffold the full FastAPI + React minimal project files (copy-paste ready).
* Export the document as PDF for submission.