End-to-End Flow (Patient: Severe Skin Rashes)

Step 1: Villager Accesses Portal

* Villager opens the telemedicine web portal in their phone’s browser (portal works in 2G/3G).
* Portal is in local language, with voice guidance for navigation.

Step 2: Symptom Reporting

* The portal shows a simple form:
  + Name, age, gender.
  + Symptoms (villager can type or speak).
  + Option to upload photo of the skin rash (image is compressed automatically for low bandwidth).
* If internet is weak:
  + Form data is saved offline in browser storage (PWA).
  + Data syncs automatically when network is available.

Step 3: Data Reaches Server

* Once connected, villager’s case is uploaded to backend server.
* Case is saved in database → tagged with *“Dermatology”*.
* Server sends a notification to available doctors.

Step 4: Doctor Reviews Case

* Doctor logs into doctor portal (dashboard).
* Doctor sees:
  + Villager’s description (“severe skin rashes, itching”).
  + Photo (low-res but enough to identify condition).
* Doctor decides treatment or asks for more details.

Step 5: Doctor Responds

* Doctor replies with:
  + Prescription (in text).
  + Additional advice (“avoid scratching, keep area clean, apply ointment”).
  + Optionally attaches an audio note in local language for easier understanding.

Step 6: Response Delivery to Villager

* If villager is online on portal → response shows up in their dashboard.
* If villager is offline → system sends:
  + SMS with doctor’s message (in local language).
  + IVR call (villager gets a phone call, automated voice reads the prescription).

Step 7: Follow-up

* Villager applies treatment.
* If condition worsens, villager reopens portal (or calls IVR) to request follow-up.
* Doctor can then suggest nearest hospital visit if urgent.

How to Implement a Low-Bandwidth Telemedicine Web Portal

1. Frontend (Portal for Villagers & Doctors)

Goals:

* Fast load on 2G (≈ 100–200 kbps).
* Minimal assets (CSS/JS < 100KB).
* Offline-ready.

Implementation:

* Use PWA (Progressive Web App):
  + Service Workers → cache pages, load even with no internet.
  + IndexedDB / LocalStorage → save patient form offline until sync.
  + Add “Install to Home Screen” → feels like an app.
* Optimize Assets:
  + Use plain HTML + CSS + minimal JS (avoid heavy libraries).
  + Compress images to WebP or JPEG ≤ 50 KB.
  + Use lazy loading (load content only when needed).
* Tech Tools:
  + React (with Next.js for SSR) or Vanilla HTML/JS.
  + Workbox (for service worker/PWA setup).
  + Web Speech API (voice input for symptoms).

2. Backend (API + Database)

Goals:

* Handle requests with small payloads.
* Work even with intermittent internet.

Implementation:

* Lightweight API (REST/GraphQL).
* Small JSON payloads (no large nested responses).
* Server-side compression:
  + Enable GZIP/Brotli on backend (reduces response size 50–80%).
* Database: MySQL/Postgres for structured records.
* Tech Tools:
  + Python: FastAPI / Flask / Django (REST).
  + Node.js: Express / NestJS.
  + Nginx/Apache with GZIP enabled.

3. Communication Fallbacks

Because sometimes even web pages won’t load well:

* SMS Gateway Integration:
  + If portal offline → villager can text symptoms to a number.
  + Example providers: Gupshup, TextLocal, Twilio.
* IVR System:
  + Villager dials a number → hears menu in local language → records symptoms.
  + Example providers: Exotel, Twilio, Asterisk (open source).
* Sync Manager:
  + If villager submits form offline, a sync daemon checks internet and pushes data later.

4. Doctor Portal

* Doctors log in via same portal (different role).
* Features optimized for low bandwidth too:
  + Simple dashboard (text + thumbnails of images).
  + Option for low-bitrate video/audio via WebRTC with adaptive quality.
  + Text prescription reply (auto-translated if needed).

5. Deployment Optimizations

* CDN (Content Delivery Network) → deliver portal faster in rural areas.
* Edge caching → static pages load from nearest location.
* AMP (Accelerated Mobile Pages) for super-fast mobile access.