Onboarding Project: Clinical Ambient Intelligence System

Author: Yulin Feng
Duration: 3 Weeks
Initiate date: Jan 8, 2026
Current version date: Jan 9, 2026

1. Executive Summary

The objective is to architect and implement a secure, real-time "Ambient Scribe" for medical consultations. The system must ingest live audio, combine it with **historical patient records**, and intelligently populate structured medical forms (SOAP notes).

Core Philosophy:

1. **Strict Privacy Isolation:** Patient data must be stored in isolated "silos." Information from Patient A must never be retrievable during Patient B's session.
2. **Hybrid Memory:** The AI must reason across two timeframes: *Long-term History* (PDFs/Past Charts) and *Short-term Context* (Live Audio).
3. **Auditability:** Every AI-generated field must be traceable to its source (Historical Record vs. Current Spoken Word).

2. Implementation Roadmap

Phase 1: Ingestion, Identification & Security

Timeline: Week 1 **Objective:** Establish a deployed microservices architecture (Frontend, ASR, Logic) and implement basic information retrieval.

- **Architecture Note (Microservices Learning):**
    - **Service A (Frontend):** React/Next.js (UI).
    - **Service B (Ingestion):** WebSocket Server (Python/Node) for handling Audio.
    - **Service C (Intelligence):** Stateless API (Lambda/FastAPI) for PII redaction and RAG.
- **Feature 1: Cloud Infrastructure Setup**
    - **Description:** Set up the cloud environment immediately to avoid "it works on

my machine" issues.
    - **Implementation:**
        - **Frontend:** Deploy a "Hello World" React page Vercel or AWS Amplify.
        - **ASR/Socket:** Container Service (AWS ECS or Google Cloud Run)
        - **Logic/PII:** AWS Lambda (Serverless) for discrete tasks.
    - **Success Criteria:** The deployed Frontend sends a string to the Ingestion Service, which passes it to the Intelligence Service, and the response is logged in the browser console over the public internet. Do not start doing feature 2 until feature 1 is done.
- **Feature 2: Resilient WebSocket Audio Streaming & Basic Real-Time UI**
    - **Description:**
        - Implement a bi-directional stream capable of handling network jitter. The system must buffer audio locally if the connection drops and re-sync automatically.
        - A minimal UI required to test Feature 2. It must connect to the WebSocket and render the scrolling text stream.
    - **Estimate: 15 - 20 Hours.**
    - **Unit Test:** Verify the automatic re-establishment and re-sync of the buffered audio state upon connection drop.
- **Feature 3: Speaker Diarization**
    - **Description:** The system must visually and logically distinguish between "Provider" and "Patient" segments in the transcript.
    - **Handling Edge Cases:** System must default to "Single Speaker" mode if no variance is detected.
    - **Test Data: Video Test.** Run the pipeline against a publicly available doctor/patient roleplay video on public video platforms like YouTube to verify real-world acoustics.
- **Feature 4: PII/PHI Redaction Layer**
    - **Description:** Implement a filter that detects sensitive entities (Names, SSNs, Dates) and masks them in the UI *before* the text is displayed, while maintaining the encrypted original for processing.
    - **Test:** Using LLM to detect if anything needs to be redacted.
- **Feature 5: Basic Information Retrieval (RAG)**
    - **Description:** Implement a simple retrieval system. When audio is transcribed, query a vector database for relevant context (e.g., "Medical History").
    - **Goal:** Establish the pipeline of Transcription -> Embedding -> Vector Search -> Context Return.
    - **Evolution:** We will refine the accuracy and "scoping" of this in Phase 2.

Phase 2: The "History-Aware" Retrieval Engine

Timeline: Week 2 **Objective:** Harden the system against data loss and implement the complex factory patterns for handling patient history vs. current session data.

- **Feature 6: History Data Ingestion (Improved RAG)**
    - **Description:** Build the "Loader" pipeline. Parse patient history documents (PDF/Text), chunk them, create embeddings, and populate the Vector DB.
    - **Outcome:** Feature 5 (from Phase 1) now actually returns useful results because the database is populated.
    - **Testing with Generated Data:** Verify the functionality of the data loader; the synthetic patient data must correspond with the previously collected video data.
- **Feature 7: Session State Persistence**
    - **Description:** Store the "Draft" state in a database (e.g., Redis or DynamoDB). If the browser refreshes, the transcript and current form state must reload instantly.
    - **Rationale:** This is treated as a "Resilience/Non-functional" requirement.
    - **Unit Test:** Simulate a browser crash; reload page; verify transcript is identical to pre-crash state.
- **Feature 8: Patient-Scoped Vector Silos**
    - **Description:** Architect a factory pattern that creates unique, isolated vector indices for the active patient.
    - **Requirement:** Distinct storage for History (Static) vs. Session (Dynamic).
    - **Unit Test:** Assert that a search query for one patient returns zero results from another patient's isolated vector index.
- **Feature 9: Pre-Visit "Snapshot" Generation**
    - **Description:** Before the recording starts, use the History Index to generate a summary of the patient's conditions. This pre-populates the "Known History" section of the form.
    - **Test with Generated Data:** Verify that the generated pre-visit summary accurately reflects the key conditions from a provided set of history documents.
- **Feature 10: "Change in Status" Detection**
    - **Description:** Implement logic to detect conflicts (e.g., Patient says "I quit smoking" vs. History says "Current Smoker").
    - **Unit Test:** Assert that the system flags a contradiction warning when generated history and live transcript data conflict on a specific status.

Phase 3: Intelligent Form Automation & Cloud Deployment

Timeline: Week 3 **Objective:** Final logic mapping, deductive reasoning, and voice command overlays.

- **Feature 11: Source Attribution**
  - **Description:** Every data point must cite its origin.
  - **UI Requirement:** Hovering over a field shows: *"Source: Medical History (2023)"* OR *"Source: Live Conversation (10:04 AM)."*
  - **Unit Test:** Assert that every auto-populated form field contains an appropriate metadata tag indicating its data source.
- **Feature 12: Deductive Form Logic**
  - **Description:** Use History to validate current symptoms (e.g., linking "leg pain" to a history of "DVT").
  - **Unit Test:** Verify that the system deductively assigns a status like 'Recurrent' to a current symptom based on a related historical condition in the generated data.
- **Feature 13: Dynamic Form Dependency**
  - **Description:** The UI must render sections conditionally. If the LLM detects "Diabetes" in the History or Conversation, a "Blood Glucose Log" section explicitly appears.
  - **Unit Test:** Assert that the form logic engine conditionally includes a dependent section in the rendered schema when the trigger condition is met in the generated data.
- **Feature 14: Auto-Population of "Review of Systems"**
  - **Description:** If a condition exists in the History but is *not* mentioned in the Live Audio, the system should prompt: *"History of Asthma not discussed. Mark as Stable?"*
  - **Unit Test:** Verify that the system generates the expected prompt for doctor review when an unmentioned historical condition is detected in the generated data.
- **Feature 15: Voice-Driven System Commands**
  - **Description:** Implement a "Command Mode." Phrases like *"System, flag this for referral"* should not be transcribed but should trigger a logic event.
  - **Unit Test:** Assert that a voice command phrase is filtered from the final note content and triggers the corresponding logic event.
- **Feature 16: Post-Visit Audit Diff**
  - **Description:** Generate a report showing the delta between the "AI Generated Note" and the "Final Doctor Edited Note" to improve future accuracy.
  - **Unit Test:** Verify the diff report correctly identifies and itemizes the changes between the generated "AI Generated Note" and the "Final Doctor Edited Note."

3. Technical Constraints & Anti-Patterns ("What Not To Do")

To ensure the system meets production standards, you are strictly prohibited from the following:

Architectural Constraints

1. **Do Not Share Vector Indices:** You **must not** store multiple patients' embeddings in a single "Global" index filtered by metadata. Each patient session must have a discrete, logically (or physically) separated storage instance.
2. **Do Not Mix Timeframes:** You **must not** merge Historical Data and Live Transcript Data into the same text chunk. They must remain distinct so the LLM knows what *happened* vs. what is *happening*.
3. **Do Not Use Blocking I/O:** Audio processing **must not** run on the main thread.

Implementation Constraints

4. **Do Not Hardcode Form Logic:** You **must not** write if/else statements in the frontend to manage form fields. The form structure and dependencies must be defined by a schema (e.g., JSON Schema) that the AI populates.
5. **Do Not Overwrite History Blindly:** The system **must not** delete a historical condition just because it wasn't mentioned in the current session.
6. **Do Not Persist Unencrypted PII:** You **must not** store raw transcripts in plain text in the database.

4. Acceptance Criteria

- **Isolation Integrity:** A unit test must prove that a query executed in "Patient A's" session returns **zero** results from "Patient B's" history documents.
- **History Integration:** The system correctly identifies a drug name mentioned effectively "it's the same one as last time" by resolving the reference against the History Index.
- **Latency:** Form updates appear on screen < 5 seconds after the relevant topic is concluded in speech.
- **Accuracy:** The system correctly distinguishes between the Doctor's voice and the Patient's voice in >90% of test cases.