WTF Flutter Engineer — 6-Hour AI-Native Assessment (100ms + Chat)

Goal: Build two Flutter apps that work together locally:

- **Guru App (Member)**
- **Trainer App**

Core modules: Auth (mock ok), real-time chat, 100ms video call scheduling + calling, session logs, basic CRM lists.
Non-negotiable: AI-Native workflow. Candidate **must** use AI for coding, debugging, tests, docs — and **prove it** via an AI Ledger.

---

0) Exam Rules (read carefully)

- **Timebox:** 6 hours hard stop.
- **Deliverables at T+6h:** GitHub repo link + 3-min demo video.
- **AI-Native:** Maintain `AI_LEDGER.md` (all prompts + where/how used + pasted outputs or links).
- **Local-first:** Must run on Android emulator/real device without cloud backend (Firebase allowed if very fast).
- **RTC: 100ms SDK mandatory** for calls. Dummy/Dev project allowed. Include minimal token server or 100ms recommended dev approach.
- **DK Persona:** Test with **Member = "DK"** (pre-seeded profile).
- **No excuses:** If a feature is missing, document why + fallback.

---

1) Repository & Project Scaffolding

- `wtf_flutter_test/`
- `├─ README.md`
- `├─ AI_LEDGER.md`
- `├─ ARCHITECTURE.md`
- `├─ DECISIONS.md              # ADRs (#1 state mgmt, #2 storage, #3 RTC strategy)`
- `├─ token_server/             # tiny 100ms token server (Node/Go/Dart - your choice)`
- `├─ shared/`
- `│   ├─ models/`
- `│   ├─ services/             # abstractions: AuthService, ChatService, CallService, LogService`
- `│   ├─ widgets/              # reusable UI`
- `│   └─ utils/                # theme, validators, extensions`
- `├─ guru_app/`
- `│   ├─ lib/`
- `│   ├─ test/`
- `│   └─ pubspec.yaml`
- `└─ trainer_app/`
- `    ├─ lib/`
- `    ├─ test/`
- `    └─ pubspec.yaml`

- **Commits:** Conventional Commits (feat:, fix:, chore:, docs:, test:, refactor:).
- **Lint:** Enable `flutter_lints`; zero warnings in final build.
- **State mgmt:** Riverpod/Bloc/Provider — explain choice in `DECISIONS.md`.

---

2) Data Model (minimum)

- `User { id, role: [trainer|member], name, email, avatarUrl?, assignedTrainerId? }`

- `Message { id, chatId, senderId, receiverId, text, createdAt, status: [sending|sent|read] }`
- `CallRequest { id, memberId, trainerId, requestedAt, scheduledFor, note, status: [pending|approved|declined|cancelled] }`
- `SessionLog { id, memberId, trainerId, startedAt, endedAt, durationSec, rating?, trainerNotes?, memberNotes? }`
- `RoomMeta { id, callRequestId, hmsRoomId, hmsRoleMember, hmsRoleTrainer }`

Storage: local (Hive/SQLite) **plus** in-memory stream layer for "live" UX. If you use Firebase for speed, that's fine; still cache locally.

---

3) UX Scenarios (must implement)

A. First-Run & Auth

- **Guru App (DK)**
  - On first run → *Onboarding* (2 slides) → *Create DK profile* (Name prefilled "DK"), choose trainer from seeded list, auto-assign.
  - Lands on **Home** with 3 cards: *Chat with Trainer*, *Schedule Call*, *My Sessions*.
- **Trainer App**
  - Seed Trainer: "Aarav (Lead Trainer)".
  - On first run → *Login* (mock) → **Home** with 4 tiles: *Members*, *Chats*, *Requests*, *Sessions*.

Acceptance:

- If app is reinstalled, onboarding shows again; otherwise, remembered login.

- Dummy avatars shown. Dark text on light BG, clear contrast.

B. Member–Trainer Chat (Real-time feel)

- **Chat List**: Recent conversations; unread count badge; last message preview; timestamp ("5m ago").
- **Conversation Screen**:
  - Bubble UI: left/right alignment with role color (Member = Blue, Trainer = Red).
  - **Typing indicator** (simulated with 400–800ms delay on other side on message send).
  - **Message status ticks**: single (sent), double (read). Mark read when screen is open.
  - **Pull to load history**, scroll to bottom on new message.
  - **Quick replies** (chips): "Got it 👍", "Can we talk at 6?", "Share plan?"
  - **Attachments (optional bonus)**: image picker; thumbnail in bubble.

Acceptance:

- Sending/receiving works across two apps when both running.
- Status changes visible; typing dot animates.
- Empty state (no chats yet) uses illustration + CTA "Say hi".

C. Schedule a Call (100ms pipeline)

- **Member (DK)** flow:
  - Screen with Calendar (next 3 days) + time slots (30-min blocks).
  - Note field (max 140 chars).
  - CTA: **Request Call** → creates `CallRequest.pending`.

    - Toast + request appears under **My Requests**: “Pending approval by Aarav”.
- **Trainer** flow:
    - **Requests** tab: list of pending with DK’s note; Approve/Decline inline.
    - On Approve → create `RoomMeta` + scheduled entry; send system message into chat: “Call approved for 6:00 PM”.
    - On Decline → reason modal; DK sees status updated.

Acceptance:

- Date/time validation (cannot pick past).
- Conflict check: slot already approved? Show error.

D. Join Video Call (100ms)

- 10 minutes before scheduled time, both see **Join Call** button in Upcoming Calls list and in Chat toolbar (small camera icon with badge).
- **Pre-join Device Check** modal: camera preview, mic/cam toggles, role auto-mapped to 100ms role.
- **In-Call UI (100ms)**:
    - Two participant tiles (grid), name labels.
    - Buttons: Mute/Unmute, Video On/Off, Flip Camera, End Call.
    - **Network resilience**: if connection blips, auto-reconnect with loader.
- **End Call**:
    - Auto write `SessionLog` with start/end/duration.
    - Post-call sheets:
        - **Member**: Rate session (1–5), optional note.

        - **Trainer**: Add quick notes; “Mark as complete”.

Acceptance:

- 100ms room is created/used. Roles enforced.
- If one user leaves, other sees state change.
- Duration captured (mock if SDK timestamp not available, else use real).

E. Session Logs & Insights

- **List** with chips: *All*, *Last 7 days*, *This Month*.
- Row shows: date, duration, rating (if any), tap → detail modal (both notes).
- **Export (bonus)**: share text summary.

Acceptance:

- Sorting by latest.
- If empty → empty state + “Schedule your first call”.

---

4) UI Requirements (pixel-level clarity)

- **Design Language:** Clean, modern, no clutter. 8pt spacing system.
- **Typography:**
    - H1 24sp, H2 20sp, Body 14–16sp.
    - Semi-bold for titles; regular for body.
- **Colors:**
    - **Trainer App:** Primary `#E50914`, neutral greys; accents minimal.

    - **Guru App:** Primary `#1769E0`, neutral greys.
    - Success `#12B76A`, Warning `#F79009`, Error `#D92D20`.
- **States:** loading skeletons, empty, error with retry CTA.
- **Components to include (ready-made or custom):**
    - AppBar with role badge (e.g., “Trainer • Aarav”).
    - Floating “+” FAB on Chat List (starts new).
    - Sticky input bar (multiline) with send icon.
    - Time chips in scheduler.
    - CTA hierarchy: Primary (filled), Secondary (outline), Tertiary (text).
- **Motion:**
    - 150–250ms transitions; slide in chat bubbles; subtle scale on button press.

---

5) 100ms Integration (must-have tasks)

- **Token Server:** small HTTP endpoint `GET /token?userId=&role=` returning 100ms auth token. Put in `token_server/` with README to run locally.
- **Room Lifecycle:**
    - On Approve: create/get room via 100ms (or dev shortcut), save `hmsRoomId`, assign roles `trainer`/`member`.
    - Pre-Join: call token server → join with role.
    - Reconnect handler & device change listener.
- **Role Permissions:**
    - `trainer`: can mute self, can end call;

    - `member`: mute self; cannot end for both (fine if SDK limits).
- **Edge Cases:** token expired (refresh), app background/foreground, network loss.

  If exact API calls differ, document your approach in `ARCHITECTURE.md` and show it working.

---

6) Quality Gates & Acceptance Tests

Manual Test Script (reviewer will run)

1. Launch **Trainer App**, login as Aarav (seeded).
2. Launch **Guru App**, onboarding DK → assigned to Aarav.
3. DK sends “Hi Coach 👋” → Trainer sees unread badge, opens chat, replies.
4. DK schedules call “today 6:00 PM”, note: “Macros review”.
5. Trainer approves; DK sees system message + Upcoming Call.
6. At +1 min (simulate now), both tap **Join Call** → camera/mic preview → connect.
7. Trainer toggles mute/video/flip; Member sees changes smoothly.
8. End call → logs created. DK rates 5★ + note; Trainer adds notes.
9. Open **Sessions** list → latest on top with rating/duration.

Pass if: All steps succeed with clean UI, no crashes, clear feedback states.

Automated/Unit (minimum)

- Message serialization/deserialization.
- Scheduler validation (no past time).
- Log duration calculation.

---

7) AI-Native Mandatory Evidence

- **AI_LEDGER.md** (structured):
    - Prompt #, Tool (ChatGPT/Gemini/Copilot…), Intent ("generate Riverpod provider for…"), Output snippet, **Commit link** where used.
    - "Debugging with AI" entries: paste error + AI steps to fix.
    - "Refactor with AI" entries: before/after summary.
- **Repo Proof:** At least 6 commits that reference AI use in body.

---

8) Observability & DX

- **In-App Debug Banner:** small floating "⋮" button opens DevPanel:
    - Env vars (masked), build info, last 20 logs (chat, RTC, schedule).
- **Logging:** Structured logs with tags `[CHAT] [RTC] [SCHEDULE] [AUTH]`.
- **Error surfacing:** Snackbars with human copy + a "Copy error" action.

---

9) Security & Secrets

- `.env.example` with placeholders.
- DO NOT hardcode live keys. Mask in logs.
- Token server uses `.env` for 100ms creds; README shows how to run.

---

10) Performance Targets

- Cold start ≤ 2.5s on emulator.
- Chat send → render on peer ≤ 400ms (simulated ok).
- RTC join time ≤ 4s on local network.
- Scroll 60fps on chat list.

---

11) UI Copy (use as-is)

- Empty Chat: “No messages yet. Start the conversation.”
- Request Sent: “Call requested. Waiting for trainer approval.”
- Approved: “Call approved for {date} {time}.”
- Declined: “Call request declined. Reason: {text}.”
- Join Prompt: “Ready to join? Check mic and camera.”
- Ended: “Session saved to your logs.”

---

12) Scoring Rubric (100 points)

- **Architecture & Code Quality (20)** — layers, naming, lint-clean, null-safety.
- **Chat UX & Reliability (15)** — statuses, typing, history, animations.
- **Scheduler & Workflow (10)** — conflict checks, UX clarity.
- **100ms Calls (25)** — join/leave, roles, reconnection, device toggles.
- **Session Logs & Ratings (10)** — completeness, filters.

- **AI-Native Proof (10)** — AI_LEDGER depth, real usage.
- **Polish & DX (10)** — error states, DevPanel, docs, demo video.

Hard Fail (0): No 100ms integration, or no AI ledger, or app doesn't run.

---

13) Submission Checklist (candidate)

- Repo builds both apps with one command in README.
- Token server runs locally; instructions + env sample.
- 100ms join works on both apps.
- Chat works both ways with read receipts + typing.
- Scheduler approve/decline; conflict handled.
- Session logs populate after call.
- AI_LEDGER.md with ≥10 meaningful entries.
- 3-min demo video (screen recording) covering end-to-end.

---

14) Reviewer Quick Notes (internal)

- Run the 9-step Manual Test.
- Skim `AI_LEDGER.md` for authenticity (copy text vs meaningful adaptation).
- Check `DECISIONS.md` for rationale maturity.
- Skim code for separation (`services/`, `providers/`, `views/`).
- Open DevPanel → review logs while performing actions.

15) Optional Stretch (only if time remains)

- Push notifications (local scheduled for reminder).
- File/image attachments in chat.
- Offline send queue for chat.
- Light/Dark theme toggle.
-