Full Technical Architecture — development-ready blueprint

Great — here's a complete, practical technical architecture that translates the unified product plan into an engineering blueprint: components, data models, APIs, integrations, deployment, security, and an actionable MVP slice. I'll keep it concrete so you (or an engineering team) can start implementing.

---

1 — High-level architecture (modules & data flow)

rust
CopyEdit

```
[Mobile App (Flutter)] <--- WebSocket/REST ---> [API Gateway / Backend Services] <--->
    |                                                                              |
    |
+--> [AI Orchestration / LLM Service]
    |                                                                              |
(LLM + prompt layer + memory)
    |                                                                              |
    |
+--> [Vision Service]
    |                                                                              |
    |
+--> [Voice Service (STT/TTS)]
    |                                                                              |
    |
+--> [Notifications Service]
    |                                                                              |
    |
+--> [Payments Service (Stripe/Razorpay)]
    |                                                                              |
```

```
    |
+--> [Real-time Comm (WebRTC Signalling)]
    |                                                          |
    +--> [Media Storage (S3/GCS)]
+--> [Database Layer: Postgres + Redis]
          (meal photos, call recordings)
(users, experts, products, orders)
```

Key interactions:

- Client sends voice/text/photo → Backend routes to appropriate service (LLM or Vision) → AI Orchestration composes response → TTS (if voice) and returns to app.
- WebRTC used for in-app calls (signalling via backend, media peer-to-peer or via TURN).
- Vector DB used for user memory & similarity search.

---

2 — Component-level design & recommended stack

Frontend

- Mobile: **Flutter** (single codebase for iOS & Android).
- Web (expert/admin dashboards): **React + Next.js** or **Flutter Web**.

Backend

- API: **Node.js (NestJS)** or **Dart Frog** (if you prefer Dart consistency).
- Real-time: **WebSocket** (Socket.io) or server-sent events for lightweight events.
- Calls/Video: **WebRTC** with signalling server (NestJS + simple signalling) + TURN server (coturn).

Datastores

- Primary relational DB: **PostgreSQL** (hosted via Supabase / RDS).
- Cache / sessions: **Redis**.
- Vector store (user memory/embedding search): **Pinecone** / **Weaviate** / **Qdrant**.
- Object storage: **S3** / **GCP Cloud Storage** for images, recordings.

AI & ML

- LLMs: **OpenAI GPT-4o / GPT-5 Thinking mini** or **Anthropic / Gemini** depending on cost/latency.
- Local/offline models: **Whisper.cpp** for on-device STT fallback if privacy mode required.
- STT: **OpenAI Whisper** (server) or cloud STT (Google, Azure) for scale.
- TTS: **OpenAI TTS**, **Amazon Polly**, or **Azure Neural TTS** (neural voices).
- Vision (meal recognition): **YOLOv8** fine-tuned on food dataset + depth/scale heuristics; optionally use cloud multimodal (Gemini Vision / OpenAI Vision) for better recognition.
- Embeddings: OpenAI embeddings or on-prem embedding model for vector DB.

Payments

- **Stripe** for global subscriptions; **Razorpay** for India/local markets; use RevenueCat for in-app subscription orchestration if needed.

DevOps / Infra

- Container runtime: **Docker**.
- Orchestration: **Cloud Run / ECS / Kubernetes** (K8s for scale).
- CDN & hosting: **Vercel** (web), **Firebase Hosting** (web), or S3+CloudFront.
- CI/CD: **GitHub Actions**.
- Monitoring: **Sentry** (errors), **Prometheus + Grafana** (metrics), **Datadog** (optional).

---

3 — AI Orchestration (detailed)

Responsibilities

- Route user inputs (text/voice/photo) to the correct model(s).
- Maintain conversation context and short/long-term memory.
- Insert safety checks, system prompts, personalization settings.
- Interface with Vision module for nutrition estimation and Recipe DB.
- Prepare succinct expert summaries.

Pipeline

1. Receive incoming request (voice/text/photo).
2. Normalize & transcribe (if voice): STT.
3. Retrieve context embeddings (vector DB) for user personalization.
4. Route:
    - Food photo → Vision service → Nutrition estimation → store audit & send to LLM.
    - Intent (text) → LLM prompt + user context → response.
5. LLM response may trigger actions:
    - Add grocery item to user checklist (DB update).
    - Schedule appointment (Appointment service).
    - Notify expert (push/email).
6. TTS (if enabled) synthesizes response audio.

7. Return to client.

Memory strategy

- Short-term context: last 10-20 messages (kept in session).
- Long-term memory: embeddings of key facts (preferences, allergies, long-term goals).
- Retention policy & UI to let user view/edit/delete memories.

---

4 — Vision & Nutrition flow

Meal photo flow

- Client uploads/takes photo.
- Mobile pre-process: allow cropping, scale reference (e.g., include a coin or plate).
- Upload to Vision service:
    - Food detection (YOLOv8) → identify items.
    - Portion estimation via size heuristics + optional depth sensors.
    - Map identified dish ingredients to nutrition DB (USDA + local extensions).
    - Return estimated calories/macros + confidence score.
- AI Health Assistant consumes result, logs meal, and replies.

Training & datasets

- Pre-train on global food datasets, augment with local cuisines.
- Continuously retrain with anonymized user-photo consented samples to improve regional recognition.

---

5 — Voice system & real-time comms

STT & TTS

- STT pipeline: prefer server-side OpenAI Whisper (better accuracy) or cloud STT for scale. For privacy mode, use on-device `whisper.cpp`.
- TTS: neural voices from OpenAI / Amazon Polly; cache common phrases for faster playback.

Calls (audio/video)

- Signalling: backend (NestJS) for WebRTC session negotiation.
- Media path: peer-to-peer where possible; fallback to TURN server.
- Limit enforcement: monitor session duration and automatically end at 20 minutes.
- Recording: disabled by default; require explicit consent. If recording permitted, store in S3 with encryption and retention policy.

---

6 — Data model (core tables) — simplified schema

Below are primary tables and important fields. Use migrations (Prisma / TypeORM / Flyway).

users

- id (uuid), name, email, password_hash, role (user/expert/provider/vendor/admin), city, country, timezone, bio, created_at, updated_at, status

user_profiles (for health seekers)

- user_id, goals (enum), height_cm, weight_kg, dob, gender, allergies (json), dietary_pref (enum), activity_level, onboarding_complete, preferences (json)

experts

- id (uuid), user_id (FK), expertise_tags (json), rating_avg, experience_years, certifications (json), availability (json), max_clients_default, client_limit_overrides, rate_per_appointment, location, status

food_providers

- id, user_id, name, menu (json or separate menu_items table), delivery_range_km, rating, city, status

menu_items

- id, provider_id, name, description, tags, price, nutrition_info (json), availability_schedule

products

- id, vendor_id, title, description, category, price, inventory_count, images, is_visible, created_at

appointments

- id, user_id, expert_id, type (video/voice/text), scheduled_at, duration_min, status (booked, completed, canceled), paid_amount, paid_via

messages

- id, conversation_id, sender_id, recipient_id, body, attachments, created_at, is_read

conversations

- id, participants (json), last_message, unread_count

meals

- id, user_id, image_url, items_detected (json), calories, protein, carbs, fats, confidence, logged_at

challenges

- id, name, goal_type, duration_days, price (nullable), participants (json), leaderboards (json)

subscriptions

- id, user_id, plan_id, start_date, end_date, status, payment_provider, provider_subscription_id

admin_audit_logs

- id, actor_id, action, details (json), timestamp

memory_embeddings

- id, user_id, embedding_vector, summary, created_at, ttl (explicit)

---

7 — API design (essential endpoints)

Use REST (+ WebSocket) or GraphQL. I'll list essential REST endpoints.

Auth

- POST /auth/signup
- POST /auth/login
- POST /auth/refresh
- POST /auth/logout

User

- GET /users/:id
- PUT /users/:id
- POST /users/:id/profile

- GET /users/:id/memories

AI Health Assistant

- POST /assistant/query — payload: { userId, inputType: [text|voice|image], input }
- POST /assistant/voiceUpload — for audio
- POST /assistant/photoUpload — meal photo

Vision

- POST /vision/analyzeMeal

Experts

- GET /experts — filters: city, specialization
- POST /experts/:id/acceptClient
- POST /experts/:id/schedule

Appointments & Calls

- POST /appointments/book
- GET /appointments/:id
- POST /webrtc/signalling — negotiate call
- POST /appointments/:id/end — server-side enforcement

Messaging

- GET /conversations/:id
- POST /conversations/:id/messages

Products & Orders

- GET /products
- POST /orders
- GET /orders/:id

Admin

- GET /admin/users
- PATCH /admin/user/:id/status
- GET /admin/stats

Payments/Webhooks

- POST /payments/checkout
- POST /payments/webhook/stripe

---

8 — Security, privacy, compliance

- Auth: JWT + refresh tokens. Use OAuth for social logins.
- Passwords: bcrypt/argon2 hashing.
- Data encryption at rest (DB & object storage) and in transit (TLS).
- Role-based access control (RBAC).
- E2E encryption for messaging/call content where feasible.
- Audit logs for privileged actions.
- Explicit consent flows for voice recording, photo storage, and sharing with experts.
- Data retention policy & easy data export/delete per GDPR.

- HIPAA considerations if you plan US healthcare integrations — isolate PHI, sign BAA with infra providers.

---

9 — Scalability & operations

- Stateless backend services with autoscaling.
- Use managed DB with read replicas for scale.
- Redis for ephemeral session & rate-limiting state.
- Vector DB hosted (Pinecone) to simplify scaling memory services.
- Use message broker (RabbitMQ / Google PubSub) for async jobs (e.g., nutrition analysis, transcription).
- Batch retraining pipelines for vision/nutrition models in Airflow or Prefect.

---

10 — Monitoring, observability & maintenance

- Error tracking: **Sentry**.
- Metrics: **Prometheus + Grafana**.
- Logs: Centralized logging (ELK stack / Datadog).
- Uptime: Health checks & alerts (PagerDuty).
- Usage analytics: **PostHog / Mixpanel** for product analytics.

---

11 — Privacy-first & offline capabilities

- Allow **privacy mode**: on-device STT (whisper.cpp) and local caching for critical parts of buddy functionality.
- Offer explicit toggle for local-only processing for photos and voice.
- Provide clear UI to view & delete “memories”.

---

12 — Third-party integrations

- Nutrition DB: **USDA FoodData Central** + local cuisine database augmentations.
- Maps/Local: Google Maps / Mapbox for locating local food providers and gyms.
- Wearables: Apple HealthKit, Google Fit, Fitbit API.
- SMS/Email: Twilio / SendGrid.
- Payment: Stripe / Razorpay.

---

13 — Data retention, backups & DR

- Daily incremental DB backups, weekly full backup.
- S3 versioning + lifecycle rules for media.
- Disaster recovery plan and RTO/RPO defined based on business needs.

---

14 — MVP scope (recommended for v1 — 8–12 week PoC)

MVP features (must-have):

- Core auth & role-based registration (common first screen + role-specific second screen).
- AI Health Assistant (text-first; voice in v1 minimal: upload audio for server transcription).
- Meal photo upload + basic nutrition estimation (server-side basic classifier).
- Basic user profile, goals, onboarding via conversational text.
- Expert registration + simple expert dashboard (client list, accept/reject).
- Messaging between user & expert (text).
- Appointments: booking, 2 free appointments/month logic, paid extra booking flow (Stripe integration).
- Products listing + simple checkout.
- Admin dashboard: user/expert/product moderation.
- Push notifications (reminders).

Defer to v2:

- Real-time voice always-on, WebRTC in-app video, advanced vision tuning, full offline local models, vector memory at scale, advanced personalization RL, multi-language TTS fine-tuning.

---

15 — Estimated milestones & teams

Team suggestions

- 1 Product Manager / Owner
- 1–2 Mobile Developers (Flutter)
- 1 Backend Developer (Node/NestJS)
- 1 Full-stack (React/Next) for dashboards
- 1 ML Engineer for vision & prompts
- 1 DevOps/Infra
- 1 QA

Phases

- Week 0–2: Detailed requirements, data model, infra prep.
- Week 3–6: Backend APIs, auth, DB, basic assistant text pipeline.
- Week 7–10: Mobile app MVP features + upload photo flow + messaging + expert dashboard.
- Week 11–14: Payments, appointments, admin, basic deployment.
- Week 15–20: Improve vision, add STT/TTS, scale & polish.