Custom Instructions – Build a Real-Time Local Services Marketplace App (ChatGPT-5 Playbook)

These instructions tell ChatGPT-5 exactly how to plan, code, test, and deploy a production-ready app that matches the following product statement:

An app where businesses can list service offerings (plumbers, lawn care, roofers, dog groomers, transportation providers, dog walkers, sitters, electricians, handymen, movers, furniture transport, etc.). Users can see a "Currently Available" section (like ride-share) to request immediate service or quotes, track provider arrival in real-time, chat/voice, and pay in-app. The app uses location services and integrates LLMs/ML, speech-to-text, and AI to maximize best/highest use.

---

0) Operating Mode & Ground Rules (for ChatGPT-5)

- Be opinionated: use the default stack below unless the user explicitly changes it.
- Ship in slices: produce working end-to-end verticals (auth → UI → API → DB → tests) per milestone.
- Always return runnable assets: include commands, env vars, seed data, migrations, CI config, and deployment steps.
- Security by default: least-privilege, PII minimization, encrypted secrets, Stripe for PCI scope reduction.
- Observability: instrument logs, metrics, traces, and error reporting from day one.
- Test first when feasible; otherwise include unit + integration tests in the same PR.
- No dead ends: every deliverable must include next steps and rollback guidance.

---

1) Product Scope & Roles

Roles

- Customer: browses categories, posts jobs, requests immediate help, tracks provider ETA, pays, rates.
- Provider: onboards (KYC + docs), sets service categories, schedule, service radius, goes Online/Offline, accepts jobs/quotes, navigates, completes work, gets paid.
- Admin: verifies providers, manages disputes/refunds, category taxonomy, pricing rules, promotions, and support.

Core Features

1. Discovery: category browsing, search, filters, map view.
2. Currently Available (real-time roster): show online providers within radius; request immediate service or instant quote.
3. Jobs & Quotes: customers post details (text/voice/photo/video), providers submit quotes or accept on-demand requests.
4. Tracking: live location/ETA (like ride-share), status updates (en-route, on-site, in-progress, completed).
5. Messaging & Voice: in-app chat + optional masked calling; speech-to-text for job creation.
6. Payments: upfront hold, progress payments, tips, refunds, Stripe Connect payouts to providers.
7. Ratings/Reviews & issue resolution.

8. LLM/ML Assist: intake structuring, category matching, quote drafting, safety redaction, job-summary, and provider ranking.
9. Notifications: push, SMS/email fallbacks, webhook retries.

---

2) Opinionated Tech Stack (Default)

- Monorepo: Turborepo (pnpm) or Nx.
- Mobile App: React Native + Expo (EAS build, OTA updates).
- Web Admin + Marketing Site: Next.js (App Router) + React Server Components.
- Backend: Node.js + TypeScript, NestJS (REST + WebSockets). Alternative acceptable: FastAPI (Python) with similar structure.
- Database: PostgreSQL + Prisma ORM, pgvector for embeddings.
- Cache/Queues: Redis + BullMQ.
- Storage: S3-compatible (AWS S3 / Cloudflare R2) for media uploads.
- Maps/Geo: Google Maps Platform (Places, Maps SDK, Distance Matrix for ETA) or Mapbox + Valhalla.
- Auth & Identity: Clerk (email/phone/social, MFA) or Auth.js + custom JWT; Stripe Connect Express for provider KYC/payouts.
- Payments: Stripe (PaymentIntents, Connect, webhooks, disputes).
- Notifications: Expo Push + FCM/APNs; email via Resend/SES; SMS via Twilio (masked numbers for privacy).
- LLM/STT: OpenAI GPT-4o/GPT-5 for LLM tasks; Whisper (API or server) / Deepgram for speech-to-text.
- Search/Analytics: PostHog (product analytics, feature flags, session replays), DB full-text search.
- Observability: Sentry (errors) + OpenTelemetry traces + structured logs (pino) shipped to a log sink.
- Infra: Docker + docker-compose for dev; Render or Fly.io for API; Vercel for Next.js; Neon/Aiven for Postgres; Upstash/Redis Cloud for Redis.
- CI/CD: GitHub Actions (lint, typecheck, tests, build, deploy). Secrets via Doppler or 1Password.

Naming: working title ServiceLink (change later).

---

3) System Architecture (High-Level)

- Clients: RN/Expo app (Customer/Provider modes) + Next.js Admin.
- API Gateway (NestJS):
- REST for CRUD & webhooks
- WebSockets for real-time presence, job events, and chat
- Background workers for geofencing, ETA refresh, webhook retries, payouts
- Postgres for relational data + pgvector for embeddings; Redis for presence, rate limits, queues.
- Stripe for payments + payouts; Twilio for optional masked calls.
- Maps for geocoding, distance/ETA; Expo for push notifications.

---

4) Data Model (Initial)

Users(id, role[customer|provider|admin], email/phone, password_hash, status, created_at)

Profiles(user_id FK, first_name, last_name, avatar_url, rating, city, state)

Providers(user_id FK, company_name, EIN/SSN_last4, service_radius_km, online:boolean, kyc_status, stripe_account_id)

ProviderDocuments(provider_id, type[license|insurance|background], url, verified_at)

Categories(id, name, slug, parent_id)

Services(id, category_id, name, base_price, unit[hour|fixed], metadata)

ProviderServices(provider_id, service_id, price_override, min_callout_fee)

ServiceAreas(provider_id, polygon_geojson)

Availability(provider_id, weekday, start_time, end_time, overrides_json)

LocationPings(provider_id, lat, lng, speed, heading, at timestamp) // ephemeral mirrored in Redis

Jobs(id, customer_id, category_id, status[draft|open|assigned|enroute|onsite|in_progress|completed|cancelled],
address, lat, lng, description, media_urls[], created_at, scheduled_for, requires_quote:boolean)

JobLineItems(job_id, title, qty, unit_price)

Quotes(id, job_id, provider_id, total, expires_at, notes, status[pending|accepted|rejected|withdrawn])

Assignments(job_id, provider_id, accepted_at, eta_seconds, start_at, complete_at)

ChatMessages(id, job_id, sender_user_id, type[text|image|system], content, created_at)

Payments(job_id, customer_id, stripe_payment_intent_id, amount, currency, status)

Payouts(provider_id, stripe_transfer_id, amount, status)

Refunds/Disputes(payment_id, amount, reason, status)

Reviews(job_id, rater_user_id, ratee_user_id, stars, text, created_at)

Notifications(user_id, kind, payload_json, delivered_at)

Embeddings(owner_type[user|provider|policy], owner_id, vector)

Add indexes: GIST on geo fields; btree on (status, created_at), (provider_id, online), (job_id).

---

5) API Design (Selected Endpoints)

Auth

- POST /auth/register, POST /auth/login, POST /auth/otp/verify
- POST /providers/onboarding (creates Stripe Connect Express link)

Discovery
- GET /categories, GET /services?category_id=, GET /providers/available?lat&lng&radius_km

Jobs & Quotes
- POST /jobs (create from text/voice/photo → LLM structuring)
- GET /jobs/:id, PATCH /jobs/:id (status transitions)
- POST /jobs/:id/quotes (provider → customer)
- POST /quotes/:id/accept
- WS: jobs:* channels for status, ETA, and chat

Tracking
- POST /providers/ping (lat, lng, speed, heading)
- GET /jobs/:id/eta (Distance Matrix or cached)

Payments
- POST /payments/intent (hold), POST /payments/capture, POST /payments/refund
- POST /webhooks/stripe (idempotent)

Reviews/Notifications
- POST /jobs/:id/review, GET /notifications

Admin
- CRUD categories/services, verify documents, manage refunds/disputes, impersonate user (read-only), feature flags

---

6) Real-Time Presence & Matching
- Providers toggle Online/Offline → presence stored in Redis with TTL; geo-grid index for fast nearby lookup.
- On customer request: build candidate set (radius, category, rating, availability) → rank by weighted score:
- distance (40%), acceptance rate (20%), rating (20%), responsiveness (10%), price (10%).
- For on-demand, send ring-group notifications (first-come accept). For quote, collect proposals until timeout.
- Refresh ETA via Distance Matrix; stream to clients over WS; gracefully degrade to polling.

---

7) LLM, STT & "Best/Highest Use" Automations
- Intake NLU: convert unstructured text/voice/photos into: category, sub-tasks, estimated time, materials list, risk flags.
- Redaction: strip PII from chat logs where not needed (phone, exact addresses inside LLM prompts).
- Quote Assistant: provider-side prompt that drafts line items, scope, exclusions, and terms; editable before sending.

- Smart Matching: LLM-assisted scoring features (normalize skills, past jobs) feeding the ranking function.
- Knowledge RAG: policies, safety, pricing playbooks → serve answers in app help and admin macros.
- Summaries: auto-produce job summaries and completion notes for receipts and support.
- Speech-to-Text: Whisper/Deepgram for job creation, in-chat voice notes, and hands-free provider updates.

Keep prompts in code with versioned prompt templates; add unit tests for critical prompt outputs.

---

8) Security, Privacy & Compliance

- RBAC by role; route guards + attribute checks on job ownership.
- PII: encrypt at rest (Postgres column encryption for name/phone/address), short retention for geodata.
- Secrets in Doppler/1Password; never commit .env.
- Stripe: keep all card data in Stripe elements/tokens; listen to webhooks with idempotency keys.
- Audit log all admin actions; immutable append-only table.
- Secure media: signed URLs; virus scan if accepting docs.
- Rate limits & abuse controls (per-IP, per-user, per-route); WAF on the edge (Vercel/Cloudflare).

---

9) Dev Experience, CI/CD & Observability

- Repo Layout

```
/ apps
  /mobile         # Expo RN app (customer & provider modes)
  /web            # Next.js admin/marketing
  /api            # NestJS (REST, WS, workers)
/ packages
  /ui             # shared RN/React components
  /config         # eslint, tsconfig, prettier
  /schemas        # zod DTOs, OpenAPI, prompt templates
/ infra
  docker-compose.yml, terraform (optional), fly.toml, render.yaml
/.github/workflows/ci.yml
```

- Scripts: pnpm dev, pnpm db:migrate, pnpm db:seed, pnpm test, pnpm lint, pnpm build.
- Tests: Vitest/Jest (unit), Playwright (E2E web), Detox (mobile), Supertest (API), Contract tests for SDKs.
- Observability: Sentry client/server, OpenTelemetry exporter; dashboard links in README.

---

10) Deployment Targets (Default)

- API: Render (autoscale) or Fly.io; attach managed Postgres + Redis; S3/R2 bucket.
- Web: Vercel (env-linked previews, edge cache).
- Mobile: Expo EAS (iOS/Android), release channels for staging/prod; OTA updates.
- DNS: Cloudflare (proxy, WAF, SSL).

Include one-click buttons (Render blueprint, Vercel deploy) and CLI commands in the deliverables.

---

11) Milestones & Acceptance Criteria

M0 – Bootstrap (Week 0)
- Turborepo scaffold with apps/packages. Docker compose for Postgres+Redis. Healthcheck route.
- Deliverables: repo ZIP, README quickstart, .env.example, CI passing.

M1 – Auth & Provider Onboarding (Week 1)
- Clerk/Auth.js login, profiles, basic RBAC.
- Stripe Connect Express onboarding link + webhooks.
- Deliverables: API routes + tests; mobile flows for sign-in, provider onboarding.

M2 – Categories, Services, Discovery (Week 2)
- CRUD categories/services; customer discovery (list/map); provider service config.
- Deliverables: seed categories, admin CRUD, map screen, tests.

M3 – Real-Time Availability & Jobs (Week 3)
- Online/Offline presence via Redis; nearby providers endpoint; create job (text/voice/photo) → LLM structuring; chat.
- Deliverables: WS channels, ETA polling; RN screens; E2E demo.

M4 – Quotes, Assignment, Tracking (Week 4)
- Provider quotes; instant accept for on-demand; live tracking + status timeline.
- Deliverables: acceptance logic, race conditions handled, retries.

M5 – Payments & Reviews (Week 5)
- PaymentIntents (hold/capture), refunds, payouts; ratings/reviews; receipts.
- Deliverables: full payment happy path + dispute flow; admin resolution tools.

M6 – LLM, STT & RAG (Week 6)
- Whisper job-intake; quote assistant; policy RAG; safety redaction.
- Deliverables: prompt templates, tests, latency budgets.

M7 – Hardening & Beta (Week 7)
- Load/perf, security review, monitoring dashboards, runbook. Pilot launch.
- Deliverables: playbooks, on-call guide, rollback plan.

Timelines are placeholders; compress/expand per scope. Each milestone must be shippable.

---

12) What to Return per Milestone (Non-Negotiable)

1. Code: full file tree with per-file diffs; schema.prisma migrations; seed scripts.
2. Infrastructure: docker-compose, deploy descriptors (Render/Vercel), worker process config.
3. Env Vars: .env.example documented (purpose, default, secret owner).
4. Docs: README quickstart + per-feature docs; OpenAPI/Swagger; Postman collection.
5. Tests: unit + integration + E2E notes (how to run locally); sample data.
6. Monitoring: Sentry DSNs wired; health endpoints; synthetic check recipe.

---

13) Sample Prompts & Checklists (Reusable)

13.1 Bootstrap Prompt (use at project start)

“Create a Turborepo monorepo with apps/mobile (Expo RN), apps/web (Next.js), apps/api (NestJS). Configure Prisma with Postgres and Redis via docker-compose. Add pnpm workspaces, ESLint/Prettier, Husky pre-commit. Generate base screens (Auth, Home, Map) and a healthcheck API route. Provide commands, .env.example, and a README.”

13.2 Feature Slice Prompt (example: Real-Time Availability)

“Implement provider Online/Offline presence. API: POST /providers/status {online:boolean}; store in Redis with TTL 90s and geo index. Endpoint GET /providers/available?lat&lng&radius_km. Mobile: toggle switch + background location ping every 30s when Online. Include WebSocket channel presence:* for roster updates and tests for staleness cleanup.”

13.3 Debugging Checklist

- Reproduce with seed data; capture logs (pino JSON) and Sentry event.
- Verify env vars and secrets; confirm webhook signatures (Stripe/Twilio).
- Check Redis TTL drift; WS backpressure; idempotency keys on mutations.
- Confirm DB migrations; check N+1 queries; inspect slow query log.

13.4 Security Review Checklist

- RBAC unit tests for each sensitive route.
- PII map, retention policy, and encrypted columns validated.
- Rate limits and CAPTCHA on auth & sensitive flows.
- Signed URLs for uploads; content-type validation.

---

14) Risks & Mitigations

- Location accuracy/ETA: use high-accuracy mode only when Online; snap-to-road; fallback to last known.
- Provider scarcity: seed supply, offer scheduled jobs & quotes; expand radius with clear ETAs.
- Fraud/chargebacks: Stripe 3DS, device fingerprinting, hold/capture until completion photos.
- LLM hallucination: constrain prompts, provide structured outputs (zod schemas), human-in-the-loop edits.

- Push reliability: WS + push + SMS fallback with exponential backoff and idempotency.

---

15) Open Questions (Assumptions until clarified)

- Target markets/regions & tax rules? (affects Stripe tax, service fees, receipts)
- Do providers need background checks beyond Stripe KYC?
- Are scheduled recurring services (e.g., weekly lawn care) in scope for v1?
- Do we support multi-provider jobs (crew assignments) at launch?
- Cancellation/rescheduling policies and fees?

---

16) Immediate Next Step (Executable)

Return in the next message:

1. Repo bootstrap (monorepo scaffolding commands and file tree),
2. DB schema.prisma (v1) with migrations,
3. API skeleton (NestJS modules: auth, providers, jobs, payments),
4. Expo starter screens (Auth, Home, Map, Availability Toggle),
5. docker-compose (Postgres, Redis), and
6. A Render + Vercel deploy plan with environment variables.

All assets must be runnable locally with pnpm i && docker compose up -d && pnpm db:migrate && pnpm dev.

This playbook is the contract: follow it to produce production-grade code and deployments for a real-world launch. Edit the stack sections freely if the user changes constraints (e.g., different cloud or auth/payment providers).