**Fix It – Technical Plan**

**1) Overview**

* Goal: Mobile app connecting users in Egypt with home-repair technicians (electrical, plumbing, carpentry, AC, etc.).
* Architecture: Clean Architecture (Presentation: Flutter+BLoC, Domain: Entities/UseCases/Repos, Data: Models/DataSources/Repo impls). DI via GetIt, networking via Dio/Retrofit, JSON serialization, local caching via shared\_prefs/sqflite, secure storage via flutter\_secure\_storage. Backend REST+PostgreSQL. FCM for notifications. Arabic (RTL) first.
* Identity: Firebase Authentication only; backend verifies Firebase ID Tokens per request via Admin SDK; no backend JWTs, no refresh endpoints.

**2) Authentication: Firebase Auth only**

* Client (Flutter): Use Firebase Auth SDK (email/password; add phone sign-in as first-class for Egypt).
* Server (Backend):
  + Every protected request includes Authorization: Bearer <Firebase ID Token> over HTTPS.
  + Verify token on each request with Firebase Admin SDK verifyIdToken; extract uid for RBAC and data ownership.
  + Optionally cache decoded tokens briefly to reduce verification overhead on hot endpoints.
  + Use custom claims only for access control flags if needed; store all other user data in DB.
* Data model: Store firebase\_uid UNIQUE on Users; email/phone optional to support phone-first onboarding for technicians.

**3) Payments: Cash only (MVP)**

* No payment gateway integration in MVP.
* UI: “Cash on completion” in booking confirmation.
* Backend:
  + Providers mark job completed; require client confirmation (two-step) to finalize cash status and reduce disputes/fraud; auto-complete after N hours with dispute\_window flag for support intervention.
* DB:
  + Payments table remains for extensibility.
  + payment\_method="cash"; transaction\_status: pending→completed (after client confirmation)→refunded (manual if needed).

**4) Proximity search (lat/lng + server-side distance filtering)**

* Storage:
  + ServiceProviders: latitude, longitude (and last\_location\_updated\_at).
  + Bookings: address\_details\_json includes latitude, longitude.
* API:
  + GET /providers?service\_id=&search\_query=&min\_rating=&available\_at=&lat=&lng=&radius\_km=&max\_price=&sort=distance|rating|price.
  + Response includes distance\_km (rounded to 0.1km) and optional ETA\_hint\_minutes (simple heuristic, e.g., 25km/h intra-city) flagged as estimate\_only.
* Query strategy:
  + Bounding box prefilter using lat/lng ranges, then Haversine to compute accurate distance, HAVING distance<=radius\_km, ORDER BY distance when requested.
  + Keep a feature flag to switch to Postgres earthdistance or PostGIS later for spatial indexing/performance if volume grows.
    - earthdistance: ll\_to\_earth + earth\_distance for simpler accurate queries.
    - PostGIS geography: ST\_Distance and spatial indexes for scalability.

**5) Core features (MVP)**

* Users:
  + Browse categories/services; list nearby technicians with ratings/prices.
  + Book appointment: service, provider, date/time, address on map, notes.
  + Track booking status: pending, confirmed, in\_progress, completed, cancelled.
  + Notifications for booking updates.
  + Rate provider after completion (verified booking).
* Providers:
  + Professional profile (profession, services, pricing, bio).
  + Coverage configuration (center and radius) and availability schedule.
  + Accept/decline jobs; update job status; view job address.
* Admin (light initially):
  + Manage categories/services.
  + KYC review and provider verification.
  + Monitor bookings, disputes, and timeouts.

**6) Data model (PostgreSQL highlights)**

* Users
  + user\_id UUID PK, firebase\_uid VARCHAR UNIQUE NOT NULL, full\_name, email UNIQUE NULL, phone\_number UNIQUE NULL, role ENUM('client','provider','admin'), profession, profile\_picture\_url, created\_at, updated\_at.[[2]](#fn2)
* ServiceCategories
  + category\_id UUID PK, name UNIQUE, icon\_url.
* Services
  + service\_id UUID PK, category\_id FK, name, description, image\_url, base\_price.
* ServiceProviders
  + provider\_id UUID PK (FK Users.user\_id), bio, average\_rating REAL default 0, total\_ratings INT default 0, is\_verified BOOLEAN default false, availability\_json JSONB, latitude REAL, longitude REAL, last\_location\_updated\_at TIMESTAMP, coverage\_center\_lat REAL, coverage\_center\_lng REAL, coverage\_radius\_km REAL, kyc\_status ENUM('pending','approved','rejected'), kyc\_documents JSONB.
  + Indexes: composite (latitude, longitude); plus individual where helpful.
* ProviderServices (M2M)
  + provider\_service\_id UUID PK, provider\_id FK, service\_id FK, price REAL, UNIQUE(provider\_id, service\_id).
* Bookings
  + booking\_id UUID PK, user\_id FK, provider\_id FK, service\_id FK, status ENUM('pending','confirmed','in\_progress','completed','cancelled','disputed'), booking\_timestamp TIMESTAMP NOT NULL, scheduled\_start\_time TIMESTAMP NOT NULL, scheduled\_end\_time TIMESTAMP NULL, address\_details\_json JSONB NOT NULL (includes latitude, longitude), total\_price REAL NOT NULL, payment\_id FK NULL, user\_rating\_id FK NULL, provider\_rating\_id FK NULL, notes TEXT.
  + Audit/flow: expires\_at TIMESTAMP, accepted\_at, in\_progress\_at, completed\_at, cancelled\_at, cancelled\_by ENUM('client','provider','system'), requires\_client\_confirmation BOOLEAN, client\_confirmed\_at TIMESTAMP.
  + Concurrency: prevent overlapping confirmed/in\_progress bookings per provider via transactional checks; consider exclusion constraints with range types later.
* Payments (cash)
  + payment\_id UUID PK, booking\_id UUID UNIQUE FK, amount REAL, currency VARCHAR(3), payment\_method ENUM('cash'), transaction\_status ENUM('pending','completed','failed','refunded'), transaction\_timestamp TIMESTAMP, external\_transaction\_id VARCHAR NULL.
* Ratings
  + rating\_id UUID PK, booking\_id FK NOT NULL, rater\_user\_id FK NOT NULL, rated\_entity\_id UUID (provider\_id), rating\_value INT CHECK 1..5, comment TEXT, rating\_timestamp default now(), verified\_booking BOOLEAN default true; UNIQUE(booking\_id, rater\_user\_id).
* Notifications
  + notification\_id UUID PK, user\_id FK, title, body, type ENUM('booking\_confirmation','review\_request','booking\_reschedule','app\_update','special\_offer','payment\_success'), timestamp default now(), is\_read BOOLEAN default false, target\_data\_json JSONB includes type, ids, version.
* AppSettings, FAQs as previously defined.

**7) Indexes (key)**

* Users: idx\_users\_firebase\_uid, idx\_users\_email, idx\_users\_phone\_number, idx\_users\_role.
* ServiceProviders: composite idx\_providers\_lat\_lng(latitude, longitude), idx\_providers\_verification, idx\_providers\_rating.
* Services: idx\_services\_category\_id, idx\_services\_name.
* ProviderServices: idx\_provider\_services\_provider\_id, idx\_provider\_services\_service\_id.
* Bookings: idx\_bookings\_user\_id, idx\_bookings\_provider\_id, idx\_bookings\_status, idx\_bookings\_scheduled\_start\_time, idx\_bookings\_expires\_at.
* Payments: idx\_payments\_booking\_id, idx\_payments\_transaction\_status.
* Ratings: idx\_ratings\_rated\_entity\_id, idx\_ratings\_rater\_user\_id.
* Notifications: idx\_notifications\_user\_id, idx\_notifications\_timestamp, idx\_notifications\_is\_read.
* Optional later: Postgres earthdistance or PostGIS indexes for geo queries.

**8) API contract (refined)**

* Auth
  + All protected endpoints require Authorization: Bearer <Firebase ID Token>; backend verifies each call via Admin SDK verifyIdToken.[[5]](#fn5)[[1]](#fn1)
  + GET /auth/me → returns current DB user resolved from token for client bootstrap convenience.
* Services
  + GET /service-categories
  + GET /services
  + GET /services/{service\_id}
* Providers
  + GET /providers?service\_id=&search\_query=&min\_rating=&available\_at=&lat=&lng=&radius\_km=&max\_price=&sort=distance|rating|price → includes distance\_km and availability\_preview.
  + GET /providers/{provider\_id}
* Bookings
  + POST /bookings → validates provider coverage and time-slot conflicts; creates booking status=pending; returns expires\_at for acceptance window.
  + PUT /bookings/{id}/accept (provider), PUT /bookings/{id}/decline (provider).
  + PUT /bookings/{id}/start (provider), PUT /bookings/{id}/complete (provider).
  + PUT /bookings/{id}/confirm-completion (client) → finalizes cash completion.
  + PUT /bookings/{id}/cancel (client/provider/admin).
  + PUT /bookings/{id}/dispute (client/provider) within window.
  + GET /users/{user\_id}/bookings?status=upcoming|past|all
  + GET /bookings/{booking\_id}
* Payments (cash flow only)
  + PUT /bookings/{id}/complete-payment-cash → internally set pending client confirmation; finalization via confirm-completion.
* Profile & settings
  + GET/PUT /users/{user\_id}/profile
  + GET/PUT /users/{user\_id}/app-settings
* Notifications
  + GET /users/{user\_id}/notifications
  + PUT /notifications/{notification\_id}/read
* Ratings
  + POST /ratings { booking\_id, rated\_entity\_id, rating\_value 1..5, comment? } → only for completed bookings (verified\_booking=true).
  + GET /providers/{provider\_id}/ratings

**9) Booking lifecycle and SLAs**

* Acceptance window: provider must accept/decline within X minutes; otherwise system auto-cancels or reassigns; notify with FCM deep links.
* Status transitions:
  + pending → confirmed (accept) → in\_progress (start) → completed (complete) → payment completed after client confirmation.
  + Any stage can move to cancelled with cancelled\_by recorded; disputes allowed within a window.
* Audit timestamps for all transitions to support support workflows and analytics.

**10) Availability and double-booking prevention**

* availability\_json schema:
  + Weekly blocks with day\_of\_week, start\_time, end\_time; exceptions list for blackout dates or special windows; backend validates schema and overlapping ranges.
* Double-book prevention:
  + Transactional query ensures no overlapping confirmed/in\_progress bookings for the provider in the requested window.
  + Consider Postgres range types and exclusion constraints once stabilizing.

**11) Provider verification, coverage, and safety**

* KYC workflow: upload national ID and skill proof; kyc\_status pending→approved/rejected; only approved providers are surfaced prominently.
* Coverage enforcement: coverage\_center\_lat/lng + coverage\_radius\_km; block bookings outside radius unless provider explicitly accepts (future: surcharge).
* is\_verified badge and filter; emergency contact field; optional trade license upload.

**12) Notifications and conventions**

* target\_data\_json includes {type: "booking" | "rating" | "offer", booking\_id?, provider\_id?, version}.
* Scheduled jobs handle acceptance timeouts and auto-completions; all changes push FCM notifications with deep-links.

**13) Security**

* HTTPS everywhere; consider SSL pinning in app later.
* Server-side token verification on every protected call; never trust client-only claims.
* Store tokens securely in flutter\_secure\_storage; do not embed secrets.
* Input validation and RBAC on server.
* Structured audit logging for booking and payment transitions (actor\_uid, IP/device if available).

**14) Performance and scalability**

* Frontend: const widgets, builder lists, image compression/lazy loading; pagination and caching.
* Backend: indexes as above; bounding-box prefilter before Haversine; optional upgrade to earthdistance or PostGIS when scale requires it.
* Token verification: consider short-lived cache of decoded tokens to reduce overhead while keeping correctness.

**15) Testing and operations**

* Unit tests: UseCases, Repositories, BLoCs with mocks; focus on booking lifecycle, availability validation, and provider search filtering.
* Widget tests for key flows; integration tests for end-to-end: auth→search→booking→accept→complete→confirm→rate.
* CI: run tests + static analysis on PRs; error logging via Crashlytics/Sentry for clients and structured logs on server.
* Admin panel (read-first, then write): KYC review, booking monitors, disputes, status overrides with audit logs.

**16) Roadmap (6-week MVP)**

* Week 1: Project setup; Firebase Auth integration; Users(firebase\_uid) model; /auth/me; base services/categories.
* Week 2: Providers listing with proximity (bounding-box + Haversine), provider profile; composite lat/lng index.[[7]](#fn7)[[8]](#fn8)
* Week 3: Booking flow; coverage and slot conflict checks; acceptance window and expires\_at.
* Week 4: Provider dashboard (accept/decline/start/complete); FCM notifications; user bookings list.
* Week 5: Ratings (verified booking); profile/settings; KYC review basics; fraud controls (two-step completion).
* Week 6: RTL QA; security hardening (token verification paths, logging); performance; pilot release.

**17) Implementation snippets (server-side proximity)**

* Bounding box parameters:
  + dLat = radius\_km/111; dLng = radius\_km/(111\*cos(radians(:lat))).
* Haversine select (km) example:
  + distance\_km = 6371\*acos(cos(radians(:lat))\*cos(radians(latitude))\*cos(radians(longitude)-radians(:lng))+sin(radians(:lat))\*sin(radians(latitude))).
* Postgres alternatives:
  + earthdistance: earth\_distance(ll\_to\_earth(:lat,:lng), ll\_to\_earth(latitude,longitude)).
  + PostGIS: ST\_Distance(geography, ST\_MakePoint(:lng,:lat)::geography) with spatial index if upgraded.