# TaxiBaba — From-Scratch Product & Engineering Blueprint (v1)

*Last updated: Aug 31, 2025*

## 1) Vision & MVP guardrails

**Goal:** Bike-taxi + courier/parcel micro-logistics for Tier-1/2 Indian cities with a flat ₹5 commission per ride.

**Primary roles** - **Rider (customer):** books rides, courier, food parcel; pays via UPI/cards; rates trip. - **Captain (bike driver):** KYC/onboarding, uploads RC/insurance/Pollution, *must* have taxi number plate; accepts jobs, earnings wallet, withdrawals. - **Ops/Admin:** dispute handling, surge/zone mgmt, fraud checks, Captain verification.

**MVP must-haves (Phase 1, 60–90 days)** - Phone OTP auth (India +91) for Rider & Captain. - Live map, pickup/drop pin, fare estimate, flat commission logic. - Captain app: go-online/offline, accept/reject, in-trip navigation. - Payments: UPI intent (GPay/PhonePe/Paytm), cash; daily payout to Captain via UPI/IMPS. - Courier/Parcel flow: size tiers (S/M/L), photos, basic Proof-of-Delivery (POD) with OTP. - Support: simple ticketing, trip chat/voice mask, SOS and live location share. - Basic analytics: trips/GMV/retention; admin web dashboard.

**Nice-to-have later**: surge, subscriptions, referral program, driver incentives, fraud ML, multi-city.

---

## 2) Architecture (high level)

**Mobile clients**: Rider app (Android first → iOS later), Captain app (Android first).

**Backend**: Java Spring Boot (microservices later; start monolith). Expose REST + WebSocket.

**Key services (start inside monolith modules)** - Auth & Accounts - Matching (dispatch) - Trip service (state machine) - Payments & Wallets - Courier service (POD, photos) - Notifications (FCM/SMS) - Compliance/KYC - Admin (RBAC, audit)

**Infra** - DB: Postgres 15 (RDS/Cloud SQL). Redis for caching/queues. - Object storage: S3 or GCS (docs, images). - Realtime: WebSocket (Spring) + Redis pub/sub. - Maps/Geo: Mapbox or Google Maps Platform (Directions, Distance Matrix, Places, Roads), plus OpenRouteService fallback. - CI/CD: GitHub Actions → Docker → Cloud Run/ECS/Kubernetes (later). - Observability: OpenTelemetry + Grafana/Tempo/Loki; Sentry for clients.

**Security & Privacy** - JWT w/ rotation; device binding. Field-level encryption for KYC. Phone number hashing. Rate limiting. Audit trails. SOS data retention controls.

---

## 3) Data model (initial tables)

**users**(id, role[RIDER|CAPTAIN|ADMIN], phone_e164, email?, name, avatar_url, status, created_at)

**captains**(user_id PK→users.id, kyc_status, dl_number, bike_plate, taxi_plate_photo_url, rc_url, insurance_url, city, online_status, rating, banned_reason?)

**riders**(user_id PK, default_payment, rating)

**trips**(id, rider_id, captain_id?, type[RIDE|COURIER], state[REQUESTED|ASSIGNED|ARRIVED|IN_PROGRESS|COMPLETED|CANCELLED], pickup_latlng, drop_latlng, pickup_addr, drop_addr, distance_m, duration_s, fare_total, commission_fee, pay_method[CASH|UPI], started_at, ended_at, cancel_reason?)

**trip_events**(id, trip_id→trips.id, event_type, payload_json, created_at)

**courier_parcels**(trip_id PK, size[S|M|L], notes, pickup_photo_url?, delivery_photo_url?, pod_code)

**wallets**(user_id, type[CAPTAIN_EARN|COMPANY], balance_paise)

**wallet_txns**(id, wallet_id, trip_id?, amount_paise, direction[CR|DR], method[UPI|IMPS|ADJUST], status, meta_json, created_at)

**payments**(id, trip_id, provider_ref, method, status, amount_paise, created_at)

**pricing_rules**(city, base_fare, per_km, per_min, commission_flat_paise=500, surge_mult=1.0, active)

**support_tickets**(id, user_id, trip_id?, category, message, status)

**admin_users**(id, phone, role, last_login, flags)

Indexes: geo (pickup_latlng, drop_latlng using PostGIS optional later), frequent queries on user_id, created_at.

---

## 4) Trip state machine (happy path)

1. **REQUESTED** (rider places request) → dispatch.
2. **ASSIGNED** (captain accepts) → navigate to pickup.
3. **ARRIVED** → rider onboard (or parcel collected) → **IN_PROGRESS**.
4. **IN_PROGRESS** → navigate to drop. Collect payment (if cash) or confirm UPI success.
5. **COMPLETED** → ledger entries (captain earnings, commission ₹5), rating prompts, receipt.

Cancellations allowed pre-arrive with fee rules.

---

## 5) REST API (v1) — outline

**Auth** - `POST /v1/auth/otp/send` {phone} - `POST /v1/auth/otp/verify` {phone, code} → {jwt, profile}

**Rider** - `POST /v1/trips/estimate` {pickup, drop, type} - `POST /v1/trips` {pickup, drop, type, pay_method} - `GET /v1/trips/{id}` - `POST /v1/trips/{id}/cancel` - `GET /v1/me` / `PATCH /v1/me`

**Captain** - `PATCH /v1/captains/me/status` {online: true|false} - `GET /v1/dispatch/queue` (WebSocket recommended) - `POST /v1/trips/{id}/accept` - `POST /v1/trips/{id}/arrived` - `POST /v1/trips/{id}/start` - `POST /v1/trips/{id}/complete` {payment_confirmation} - `GET /v1/wallet/summary` / `POST /v1/wallet/withdraw` {amount}

**Courier** - `POST /v1/trips/{id}/courier/pod` {code} - `POST /v1/trips/{id}/courier/photo` (upload URLs)

**Admin** - `GET /v1/admin/trips?filters...` - `POST /v1/admin/captains/{id}/verify` - `POST /v1/admin/price-rules` (city surge etc.)

---

## 6) Matching / Dispatch (Phase-1 heuristic)

- Filter online captains within 3–5 km, same city.
- Rank by ETA (Distance Matrix), acceptance rate, cancellation rate, idle time.
- Assign to top candidate; offer timeout 15–20s; fall back to next.
- Use Redis sorted sets for geo buckets. Upgrade later to H3 geohash + priority queues.

---

## 7) Fare & Commission

**Fare = base_fare + (per_km * distance) + (per_min * duration) * surge_mult** - Commission fixed: **₹5** per completed trip to company wallet. - Rounding to nearest ₹1; min fare per city. - Courier: add size tier surcharge (e.g., S ₹0, M ₹10, L ₹20) and POD service fee if needed.

---

## 8) Mobile app — screens (Android first)

**Rider App** - Onboarding → OTP → Home Map → Set pickup/drop → Fare estimate → Search Captains → In-Trip → Payment → Rate → Trips. - Courier extension: add parcel size/photos and delivery OTP.

**Captain App** - Onboarding → KYC upload → Verify pending → Home (go online) → Incoming job card → Navigation → Trip controls → Earnings.

**Tech choice** - **Flutter** (single codebase, great maps & performance) *or* **React Native** (JS skill reuse). Start with **Flutter** unless team is deep in RN.

---

## 9) Dev environment & repo layout

```
/TaxiBaba
  /backend
    /app (Spring Boot)
    /modules (auth, trips, dispatch, payments)
    /migrations (Flyway)
    Dockerfile
  /mobile-rider (Flutter)
  /mobile-captain (Flutter)
  /admin-web (React + Vite + shadcn/ui)
  /infra
    docker-compose.yml (postgres, redis, localstack)
    terraform/ (VPC, RDS, S3, CloudFront)
  /docs (API, ADRs, state machines)
```

**Local quickstart** - `docker compose up -d postgres redis localstack` - `./gradlew bootRun` (backend) - `flutter run -d android` (apps)

---

## 10) Spring Boot skeleton (pseudo)

- Spring Web, Spring Security, Spring Data JPA, Validation, WebSocket, Flyway, Lombok.
- Controllers: `AuthController`, `TripController`, `CaptainController`, `CourierController`, `WalletController`, `AdminController`.
- Services: `OtpService`, `DispatchService`, `TripService`, `PaymentService`, `WalletService`.
- Entities match tables; DTOs with validation; MapStruct for mapping.

---

## 11) Payments & Payouts

- Rider pays: UPI intent flow first (no PG integration needed at MVP); verify via Deep Link callback or manual confirm.
- Captain payouts: daily net earnings → UPI collect or Payout API (RazorpayX/Paytm for Business) later.
- Ledger integrity: every trip closes with two wallet txns (Captain CR, Company CR ₹5) and optional payment record.

---

## 12) Notifications & Comms

- FCM for push. In-app chat via WebSocket (persist to trip_events). Mask phone numbers via IVR bridge later.

---

## 13) KYC & Compliance (India)

- Keep soft-KYC at MVP: DL + selfie + plate photo + RC.
- Store only necessary PII, encrypt at rest, signed URLs for S3; strict RBAC in Admin.
- Safety: SOS sends live lat/long + trip id to a trusted contact and Ops inbox; privacy policy/ToS published.

---

## 14) Analytics (MVP)

- Event log: app_open, request, accept, start, complete, cancel, payment_success.
- Daily dashboards: DAU/MAU, trips, completion %, cancellations, avg ETA, Captain online time, GMV, take rate.

---

## 15) Testing Strategy

- Unit tests for pricing, dispatch ranking, trip state transitions.
- Contract tests for APIs (OpenAPI + Spring MockMvc).
- Load test dispatch endpoints (k6) for 500 RPS.
- Beta ring: internal + 50 Captains.

---

## 16) Rollout plan (0–90 days)

**Day 0–7 (Sprint 0)** – Repos, CI, auth/OTP mock, base schemas, map SDKs in apps.

**Week 2–3** – Trip estimate, create request, captain online/accept, state machine to complete; local cash payments.

**Week 4–5** – Courier add-ons (photos, POD OTP), wallet ledger, ₹5 commission, basic admin dashboard.

**Week 6–7** – UPI intent, live location share, SOS, support tickets; pilot in 1 zone.

**Week 8–9** – Hardening, analytics, compliance pass, beta expansion, feedback loop.

---

## 17) Launch checklist (pilot city)

- [ ] Rider & Captain Android builds on internal track.
- [ ] 100 verified Captains (taxi plates checked).
- [ ] Payment dry runs (UPI), cash handling SOP.
- [ ] On-call rotation, runbooks, incident comms.
- [ ] Legal docs (ToS, Privacy Policy), branding basics.

---

## 18) Backlog ideas

- Driver incentives (streaks, quests).
- Rider subscriptions (Prime: lower fares, priority).
- Surge + zones heatmap.
- Fraud checks (emulators, rooted device, velocity rules).
- Multilingual (EN/HI/TE/TN/KA/MR/BN).

---

## 19) Next deliverables I can generate for you

- Spring Boot starter with Flyway, JWT, PostgreSQL, Redis, WebSocket.
- DB migration SQL for all tables above.
- OpenAPI 3.0 spec (YAML) for v1 APIs.
- Flutter screen stubs (Rider & Captain) with navigation + FCM wiring.
- Basic admin dashboard (React) with trips table and map.

---

**Notes**: Keep the monolith clean with modules and clear boundaries. Defer complex PG/UPI integrations, surge pricing, and fancy ML until you've shipped a reliable pilot.