User Story — Rider-Side Booking Dev Spec

0. User Story Explanation

Feature: Core Ride Booking

User Story: As a rider, I want to enter my pickup and destination, request a ride, track my driver, and complete payment so that I can get from point A to point B using the app.

Explanation: This is the minimum viable rider experience: secure login, address entry (map/geocode), fare quote, ride request & dispatch, live driver tracking, trip lifecycle to drop-off, and payment (auth → capture) with receipts. The flow must be resilient to mobile/network churn, GPS drift, retries (idempotency keys), and protect user privacy with strict data minimization.

1. Header

Document: Rider-Side Booking — Development Specification

Label Prefix (feature): RB (used across modules/components/classes).

Version History

- v1.0 (2025-09-24) — Initial draft

Authors & Roles (never delete anyone; version-specific noted)

- Watson Chao — Feature Owner (v1.0)

2. Architecture Diagram

Rationale:

- Separation of concerns: client renders UX & streams rider location; backend owns quoting, dispatch, trip state, and payments; third-party providers (maps, payments, push) are isolated behind clients to reduce blast radius and improve testability.

- Optionality preserved: cancellation and re-quote paths never block booking; tracking runs over a dedicated realtime channel.

- Explicit information flows: label edges like quote, assignDriver, driverLocation, charge to make contracts and security reviews unambiguous.

Legend:

- **Component / Module grouping:**

  - **Client:** `BookingPageComponent, DriverTrackingComponent`, `PaymentComponent`.

  - **Backend services (modules):** `BookingModule, DispatchModule, TripsModule, PaymentsModule, IdentityModule, RealtimeGateway`.

  - **ExternalCloudServices** groups third-party systems (Maps/Geocoder SDKs, Payment Processor, Push).

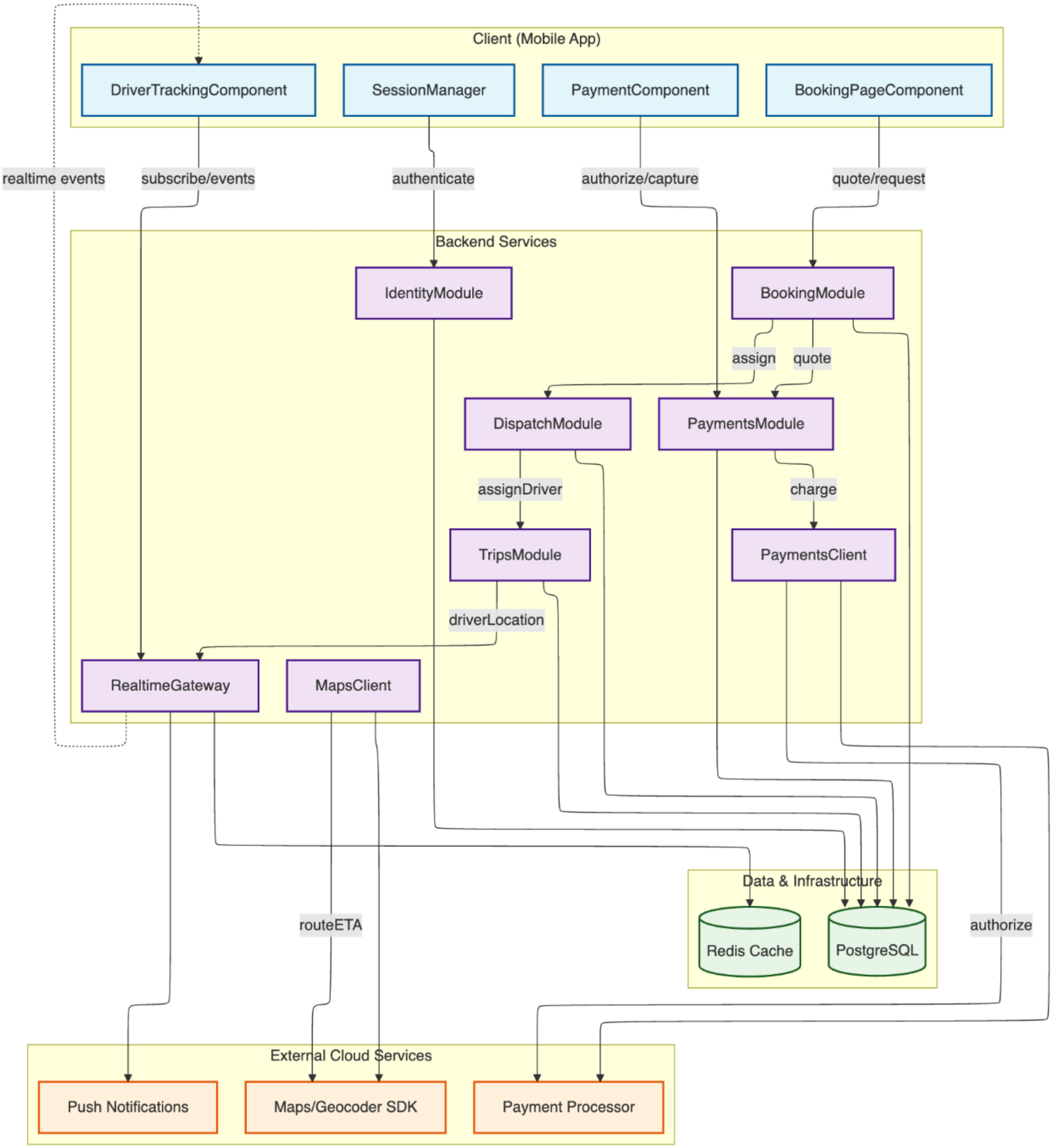
  - **DataAndInfrastructure** groups databases and caches.

  - Composition arrows ***--** indicate containment/ownership (what “lives inside” each component/module box).

- **Classes (one box per class):** each lists **Fields** (with types) and **Methods** (end with (), return type optional when void). Example: `startRealtime();` `authorizePayment()`.

- **Types shown in fields:** prefer domain types (e.g., **UniversallyUniqueIdentifier, Geolocation, MoneyAmount, Timestamp, StructuredDocument**). If a rare type appears, provide a quick example (e.g., Geolocation example: latitude 40.44, longitude −79.95).

- **Arrows between classes:**
  - `-->` = uses/calls (info or control flow)
  - `*--` = composition/ownership
  - arrow labels describe interactions (e.g., `routeETA, charge`).



3. Class Diagram

Rationale:

- One box per class with typed fields & () methods to enforce clarity and support code generation and implementation checklists.

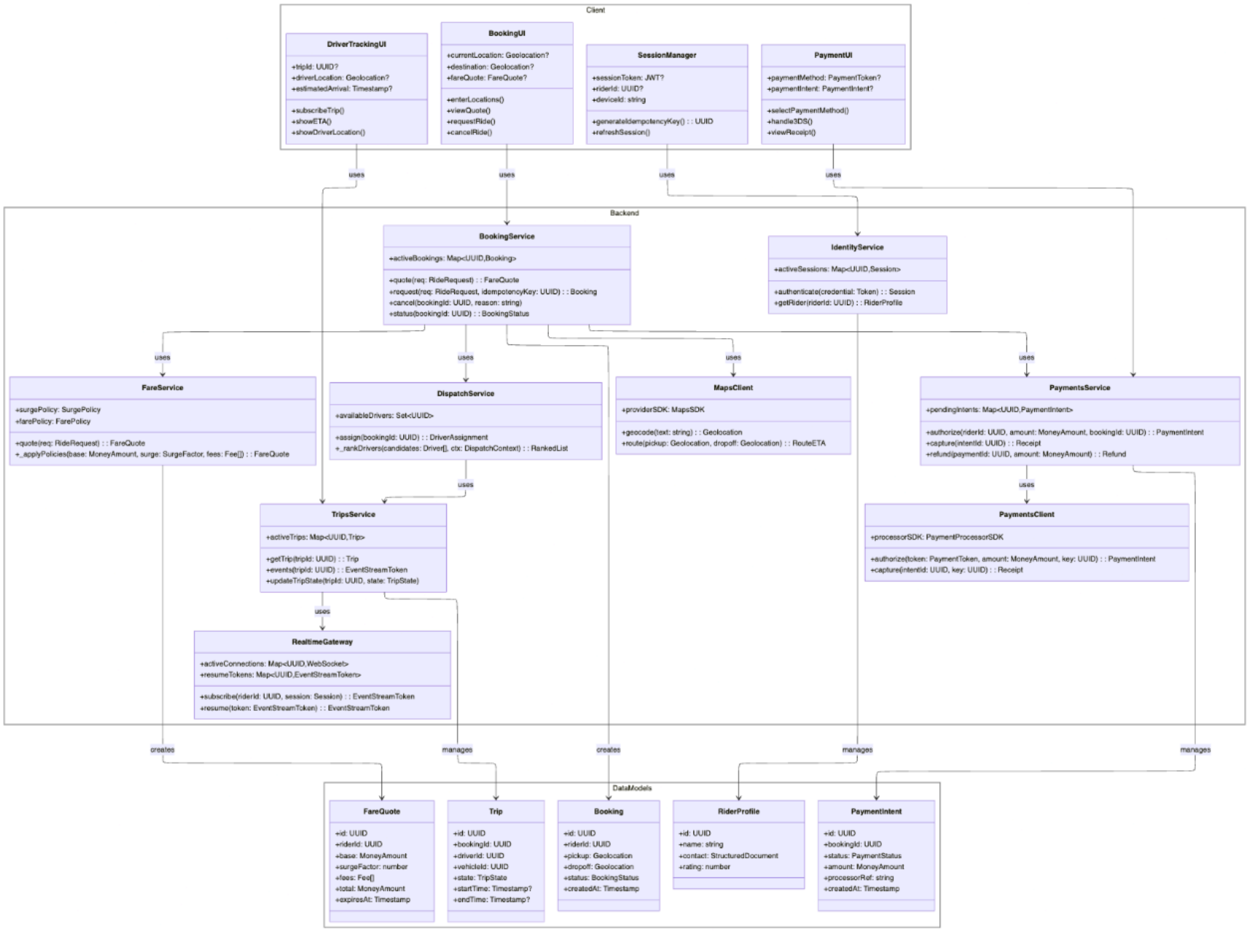
- Policy objects for fare rules (e.g., FarePolicy, SurgePolicy) allow product iteration without touching orchestration code.

- Provider client boundaries (MapsClient, PaymentsClient) encapsulate third-party quirks and enable verification/fraud checks in tests.

Legend:

- Class box: plain name; one box per class. Fields: fieldName: Type (Type? = nullable). Methods: methodName(params): ReturnType (return omitted if void).

- Grouping: Components/Modules own classes via *-- (composition = “inside the box”). Relations: --> = uses/calls; Service → Data implies persistence/cache.


Client
DriverTrackingUI
+tripId: UUID?
+driverLocation: Geolocation?
+estimatedArrival: Timestamp?
+subscribeTrip()
+showETA()
+showDriverLocation()
BookingUI
+currentLocation: Geolocation?
+destination: Geolocation?
+fareQuote: FareQuote?
+enterLocations()
+viewQuote()
+requestRide()
+cancelRide()
SessionManager
+sessionToken: JWT?
+riderId: UUID?
+deviceId: string
+generateIdempotencyKey() : : UUID
+refreshSession()
PaymentUI
+paymentMethod: PaymentToken?
+paymentIntent: PaymentIntent?
+selectPaymentMethod()
+handle3DS()
+viewReceipt()
uses
uses
uses
uses
Backend
BookingService
+activeBookings: Map<UUID,Booking>
+quote(req: RideRequest) : : FareQuote
+request(req: RideRequest, idempotencyKey: UUID) : : Booking
+cancel(bookingId: UUID, reason: string)
+status(bookingId: UUID) : : BookingStatus
IdentityService
+activeSessions: Map<UUID,Session>
+authenticate(credential: Token) : : Session
+getRider(riderId: UUID) : : RiderProfile
uses
uses
uses
uses
FareService
+surgePolicy: SurgePolicy
+farePolicy: FarePolicy
+quote(req: RideRequest) : : FareQuote
+_applyPolicies(base: MoneyAmount, surge: SurgeFactor, fees: Fee[]) : : FareQuote
DispatchService
+availableDrivers: Set<UUID>
+assign(bookingId: UUID) : : DriverAssignment
+_rankDrivers(candidates: Driver[], ctx: DispatchContext) : : RankedList
MapsClient
+providerSDK: MapsSDK
+geocode(text: string) : : Geolocation
+route(pickup: Geolocation, dropoff: Geolocation) : : RouteETA
PaymentsService
+pendingIntents: Map<UUID,PaymentIntent>
+authorize(riderId: UUID, amount: MoneyAmount, bookingId: UUID) : : PaymentIntent
+capture(intentId: UUID) : : Receipt
+refund(paymentId: UUID, amount: MoneyAmount) : : Refund
uses
uses
TripsService
+activeTrips: Map<UUID,Trip>
+getTrip(tripId: UUID) : : Trip
+events(tripId: UUID) : : EventStreamToken
+updateTripState(tripId: UUID, state: TripState)
PaymentsClient
+processorSDK: PaymentProcessorSDK
+authorize(token: PaymentToken, amount: MoneyAmount, key: UUID) : : PaymentIntent
+capture(intentId: UUID, key: UUID) : : Receipt
uses
RealtimeGateway
+activeConnections: Map<UUID,WebSocket>
+resumeTokens: Map<UUID,EventStreamToken>
+subscribe(riderId: UUID, session: Session) : : EventStreamToken
+resume(token: EventStreamToken) : : EventStreamToken
creates
manages
creates
manages
manages
DataModels
FareQuote
+id: UUID
+riderId: UUID
+base: MoneyAmount
+surgeFactor: number
+fees: Fee[]
+total: MoneyAmount
+expiresAt: Timestamp
Trip
+id: UUID
+bookingId: UUID
+driverId: UUID
+vehicleId: UUID
+state: TripState
+startTime: Timestamp?
+endTime: Timestamp?
Booking
+id: UUID
+riderId: UUID
+pickup: Geolocation
+dropoff: Geolocation
+status: BookingStatus
+createdAt: Timestamp
RiderProfile
+id: UUID
+name: string
+contact: StructuredDocument
+rating: number
PaymentIntent
+id: UUID
+bookingId: UUID
+status: PaymentStatus
+amount: MoneyAmount
+processorRef: string
+createdAt: Timestamp

4. List of Classes

M1 Client

- **RB1.1 BookingUI** — Pickup/Destination entry; shows route preview & fare quote; initiates request/cancel.
- **RB1.2 DriverTrackingUI** — Shows driver identity, vehicle, ETA, live location; SOS/help entry points.
- **RB1.3 PaymentUI** — Payment method selection (tokenized), SCA/3DS flows, receipt view.
- **RB1.4 SessionManager** — Handles auth/session tokens, idempotency key generation.

M2 Backend

- **RB2.1 BookingService** — Orchestrates quote → request → assign; booking lifecycle (pre-trip).
- **RB2.2 DispatchService** — Ranks/selects driver by proximity/availability/trust score; anti-double-assign.
- **RB2.3 TripsService** — Trip state machine (DriverEnRoute → Arrived → InTrip → Completed/Cancelled).
- **RB2.4 FareService** — Base fare, distance/time pricing, surge & fees policies; produces FareQuote.
- **RB2.5 PaymentsService** — Authorize/capture/refund; receipts; idempotent operations.
- **RB2.6 IdentityService** — Rider auth, device binding, minimal profile (for KYC/AML when applicable).
- **RB2.7 RealtimeGateway** — WS/SSE for assignment/ETA/driver pings with resume tokens.
- **RB2.8 MapsClient** — Geocode/reverse-geocode, route & ETA abstraction; provider shielding.
- **RB2.9 PaymentsClient** — Processor SDK wrapper (tokenization, 3DS/SCA, retries).

M4 Data

- **RB.DB.Booking** — Booking data (struct).
- **RB.DB.Trip** — Trip runtime/persisted data (struct).
- **RB.DB.FareQuote** — Quoted fare with expiry (struct).
- **RB.DB.PaymentIntent** — Authorization/capture lineage (struct).
- **RB.DB.RiderProfile / DriverProfile** — Minimal identity/contact (struct).
- **RB.DB.Vehicle** — Vehicle metadata for ETA and compliance.

5. State Diagrams

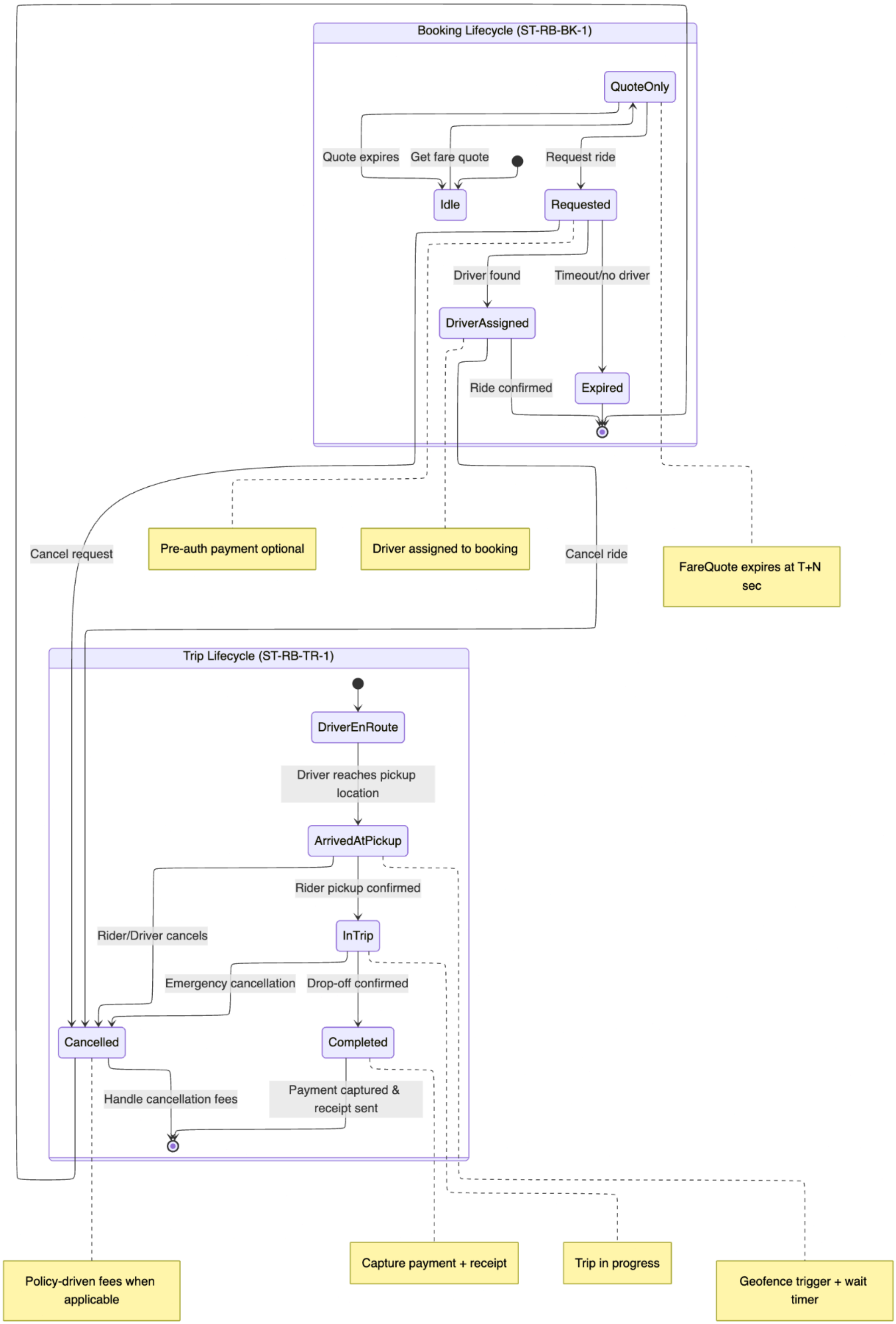
Two focused lifecycles (booking & trip), consistent with template brevity.

5a) Booking lifecycle (label: ST-RB-BK-1)

- **Idle** → **QuoteOnly** (has FareQuote expiring at T+N sec) → **Requested** (ride requested; pre-auth optional) → **DriverAssigned** → **Cancelled/Expired** (idempotent finalization).

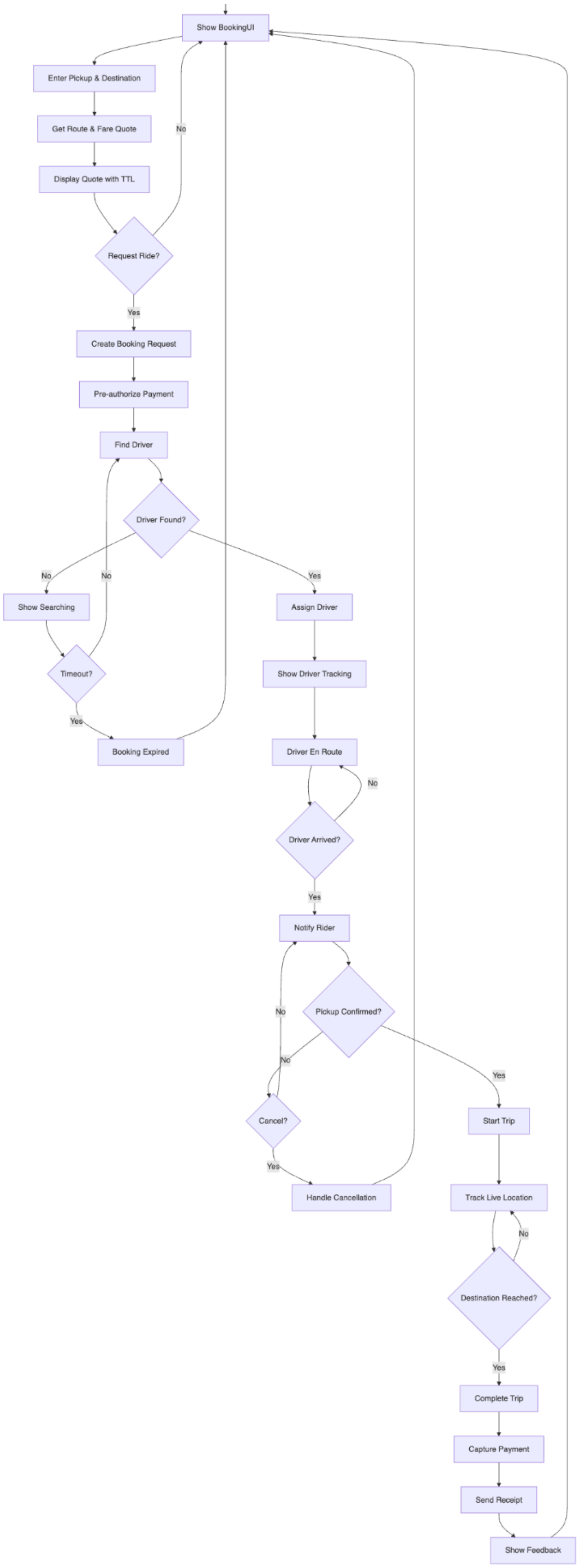
5b) Trip lifecycle (label: ST-RB-TR-1)

- **DriverEnRoute** → **ArrivedAtPickup** (geofence; wait timer) → **InTrip** → **Completed** (capture payment; receipt) / **Cancelled** (policy-driven fees when applicable).


Booking Lifecycle (ST-RB-BK-1)
QuoteOnly
Quote expires
Get fare quote
Request ride
Idle
Requested
Driver found
Timeout/no driver
DriverAssigned
Ride confirmed
Expired
Cancel request
Pre-auth payment optional
Driver assigned to booking
Cancel ride
FareQuote expires at T+N sec
Trip Lifecycle (ST-RB-TR-1)
DriverEnRoute
Driver reaches pickup location
ArrivedAtPickup
Rider pickup confirmed
Rider/Driver cancels
InTrip
Emergency cancellation
Drop-off confirmed
Cancelled
Completed
Handle cancellation fees
Payment captured & receipt sent
Policy-driven fees when applicable
Capture payment + receipt
Trip in progress
Geofence trigger + wait timer

6. Flow Chart

- Authenticate (IdentityService) → show **BookingUI**.
- Enter pickup/destination (MapsClient geocode/route) → **FareService.quote()** → display FareQuote (with TTL).
- Rider taps **Request** → **BookingService.request(idempotencyKey)** → **PaymentsService.authorize** (pre-auth) → **DispatchService.assign**.
- **RealtimeGateway** pushes **DriverAssigned** → **DriverTrackingUI** shows ETA, driver, car plate, contact options.
- Trip progression via **TripsService** + **RealtimeGateway** events; rider sees live route & ETA.
- Drop-off → **PaymentsService.capture** → receipt issued → feedback prompt.


Show BookingUI
Enter Pickup & Destination
Get Route & Fare Quote
Display Quote with TTL
Request Ride?
No
Yes
Create Booking Request
Pre-authorize Payment
Find Driver
Driver Found?
No
No
Yes
Show Searching
Assign Driver
Timeout?
Show Driver Tracking
Yes
Booking Expired
Driver En Route
No
Driver Arrived?
Yes
Notify Rider
Pickup Confirmed?
No
No
Yes
Cancel?
Start Trip
Yes
Handle Cancellation
Track Live Location
No
Destination Reached?
Yes
Complete Trip
Capture Payment
Send Receipt
Show Feedback

7. Development Risks and Failures

Runtime

- Maps provider outage → degrade to cached geocodes; block request if no safe route; user-facing error.
- Driver location gaps → extrapolate short gaps; mark stale after TTL; notify rider.
- Dispatch spikes → autoscale, queue requests, and return “assign-ETA”.

Connectivity

- Rider offline during request → local queue + retry with idempotency key; offline banner.
- WebSocket drops → reconnect with resume token; long-poll fallback.

Hardware/Config

- Node/AZ failure → multi-AZ, health checks, rolling deploys.
- Surge/price config error → schema-validated policies; instant feature-flag rollback.

Intruder/Security

- Location spoofing → route/speed sanity checks; abuse device attestation.
- Payment fraud → tokenized methods, 3DS/SCA, velocity/anomaly rules.
- Account takeover → MFA opt-in, device binding, risk scoring.

8. Technology Stack

- **TECH-MOB-1** React Native 0.74 — Mobile UI for booking & tracking.
- **TECH-WEB-1** TypeScript 5.x — Static typing across client/server; safety.
- **TECH-BE-1** Node.js 20 — Backend services; async I/O; mature tooling.
- **TECH-API-1** OpenAPI 3.1 — Contract-first APIs; codegen.
- **TECH-DB-1** PostgreSQL 16 — Relational integrity for bookings/trips/payments.
- **TECH-CACHE-1** Redis 7 — Idempotency keys, WS tokens, short-lived quotes/TTL.
- **TECH-RT-1** WebSocket/SSE — Realtime trip/assignment events.
- **TECH-MAPS-1** Maps/Geocoder SDK — Route/ETA; offline tile cache optional.
- **TECH-PAY-1** Payment Processor SDK — Tokenization, 3DS/SCA; idempotent capture.
- **TECH-OBS-1** OpenTelemetry — Traces/metrics for booking & trip funnels.
- **TECH-SEC-1** JWT/OAuth2 — AuthN/Z for mobile→backend.

Rationale: Type safety end-to-end; Postgres+Redis split; standards-based telemetry—mirrors the template's reasoning.

9. APIs

M2.Backend.RB2.1 BookingService (Public)

- quote(req: RideRequest): FareQuote
- request(req: RideRequest, idempotencyKey: UUID): Booking
- cancel(bookingId: UUID, reason: string): void
- status(bookingId: UUID): BookingStatus

M2.Backend.RB2.2 DispatchService (Private)

- assign(bookingId: UUID): DriverAssignment
- _rankDrivers(candidates: Driver[], ctx: DispatchContext): RankedList

M2.Backend.RB2.3 TripsService (Public)

- getTrip(tripId: UUID): Trip
- events(tripId: UUID): EventStreamToken

M2.Backend.RB2.4 FareService (Private)

- quote(req: RideRequest): FareQuote
- _applyPolicies(base: MoneyAmount, surge: SurgeFactor, fees: Fee[]): FareQuote

M2.Backend.RB2.5 PaymentsService (Public)

- authorize(riderId: UUID, amount: MoneyAmount, bookingId: UUID): PaymentIntent
- capture(intentId: UUID): Receipt
- refund(paymentId: UUID, amount: MoneyAmount): Refund

M2.Backend.RB2.6 IdentityService (Public)

- authenticate(credential: Token): Session
- getRider(riderId: UUID): RiderProfile

M2.Backend.RB2.7 RealtimeGateway (Public)

- subscribe(riderId: UUID, session: Session): EventStreamToken
- resume(token: EventStreamToken): EventStreamToken

M2.Backend.RB2.8 MapsClient (Private)

- geocode(text: string): Geolocation
- route(pickup: Geolocation, dropoff: Geolocation): RouteETA

M2.Backend.RB2.9 PaymentsClient (Private)

- authorize(token: PaymentToken, amount: MoneyAmount, key: UUID): PaymentIntent
- capture(intentId: UUID, key: UUID): Receipt

M1.Client.RB1.1 BookingUI (Public to app)

- enterLocations(): void, viewQuote(): void, requestRide(): void, cancelRide(): void

M1.Client.RB1.2 DriverTrackingUI (Public to app)

- subscribeTrip(): void, showETA(): void, showDriverLocation(): void

(API structure/notation mirrors your template’s patterns.)

10. Public Interfaces

Within the same component (App) — BookingUI.requestRide(), DriverTrackingUI.subscribeTrip()

Across components in the same module (Backend) — BookingService.request() uses DispatchService.assign()

Across modules

- Client→Backend: quote, request, cancel, status, subscribe/resume events
- Backend→Maps: geocode, route, ETA
- Backend→Payments: authorize, capture

Multi-interface access: Mobile SDK (TypeScript) and REST (JSON). **Example REST:**

```
POST /v1/quotes               { pickup, dropoff } → 200 { quoteId, amount, expiresAt }
POST /v1/bookings              { quoteId, paymentMethodId }  (Idempotency-Key) → 201 { bookingId }
GET  /v1/bookings/{id}         → 200 { status, assignment?, eta? }
POST /v1/bookings/{id}/cancel  { reason } → 204
WS   /v1/realtime?session=...  → EVENT { type, data }
```

(Interface framing and example-REST placement follow your template.)

11. Data Schemas

DB1 Bookings (owned by BookingService)

- id UUID PK, rider_id UUID, pickup GEOGRAPHY, dropoff GEOGRAPHY, quote_id UUID,
- status ENUM('Quoted','Requested','Assigned','Cancelled'), created_at TIMESTAMPTZ
- **Estimate:** ~160B/row (+ GEO ~48B)

DB2 Trips (owned by TripsService)

- id UUID PK, booking_id UUID FK, driver_id UUID, vehicle_id UUID,
- state ENUM('DriverEnRoute','Arrived','InTrip','Completed','Cancelled'),
- start_time TIMESTAMPTZ NULL, end_time TIMESTAMPTZ NULL
- **Estimate:** ~180B/row

DB3 FareQuotes (owned by FareService)

- id UUID PK, rider_id UUID, base MONEY, surge_factor NUMERIC(3,2), fees JSONB,
- total MONEY, expires_at TIMESTAMPTZ
- **Estimate:** ~140B/row

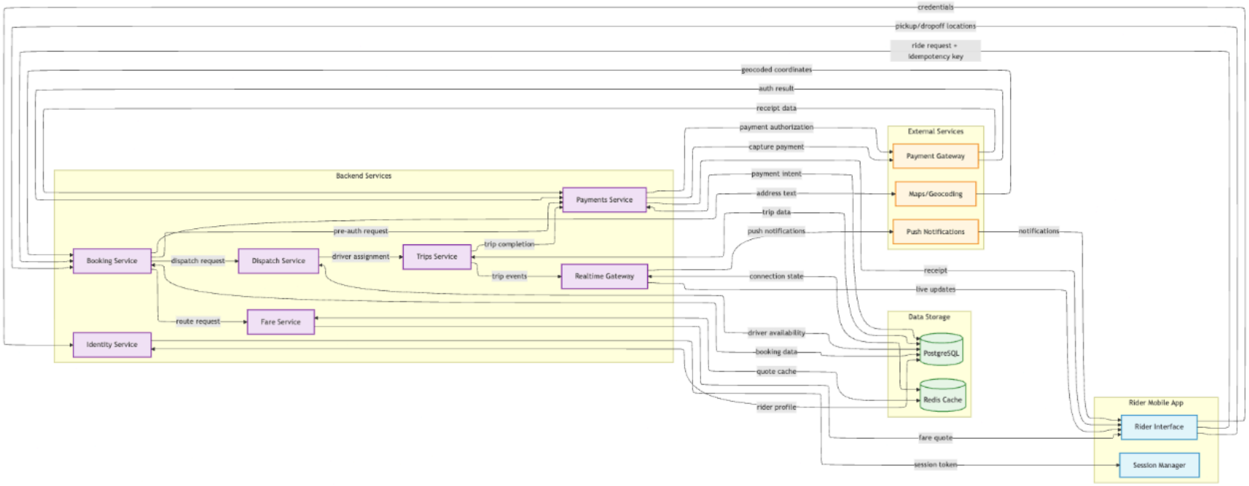
DB4 PaymentIntents (owned by PaymentsService)

- id UUID PK, booking_id UUID FK, status ENUM('Authorized','Captured','Refunded','Failed'),
- amount MONEY, processor_ref TEXT, created_at TIMESTAMPTZ
- **Estimate:** ~140B/row

DB5 RiderProfiles / DriverProfiles (owned by IdentityService)

- id UUID PK, name TEXT, contact JSONB, rating NUMERIC(2,1)
- **Estimate:** ~120B/row

Rationale: Ownership annotated per table to reduce ambiguous writes; sizing estimates to guide capacity planning & DSR costs—same style as your template’s Data Schemas + Rationale.



12. Security and Privacy

PII (temporary): auth tokens, geolocations (pickup/dropoff + driver GPS), payment tokens (non-PAN), device ID; retained only for session/trip.

Ingress: Mobile → quote/request; RealtimeGateway.subscribe.

Use: Dispatch/Trips consume rider & driver location; Payments uses tokenized methods.

Egress/Disposal: GPS logs rounded or deleted after a short retention window; quotes expire quickly; tokens never logged; receipts retained as required.

Protection: TLS 1.3, JWT access tokens, short-lived realtime tokens (Redis TTL), role-scoped DB access, audit logs, rate limits; memory-safety via types.

13. Risks to Completion

- **Maps/Geocoder variance across regions** — abstract & test; offline tiles for common corridors.
- **Dispatch fairness under surge** — simulation & SLAs; backoff on retries to avoid thrash.
- **Payments (3DS/SCA, partial captures, tips)** — explicit models; sandbox contract tests.
- **Realtime reliability over flaky networks** — resume tokens, heartbeats, backpressure.
- **Privacy & retention** — document deletion windows & DSR workflow before GA.
- **Idempotency/concurrency races** — unique keys & transactional guards; chaos tests.

(Risk list depth and style aligned with template's completion-risk section.)

14. GPT log history

https://chatgpt.com/share/68d4b7b5-9108-8011-8b78-a50dda65173d

https://claude.ai/share/2a4d713b-0c54-4db6-90e0-c65cf9d86411