

Safe2Go Global Mobility Data & Journey Engine Blueprint

Oct 6, 2026 · @VIPUL AGRAWAL

Executive summary

Safe2Go should own every static and semi-static mobility dataset itself and treat all third-party APIs as optional live enrichment. Routing must never fail because AeroAPI, a ferry B2B feed, RedBus, Mappls or Overpass is down, rate-limited or unpaid.

This document merges four inputs (a GTFS data-sourcing chat, two API-independence architecture reviews, and the Safe2Go "Logic, Databases & APIs" inventory of 2026-09-26), corrects URLs and claims that do not hold up, and adds what was missing: a concrete universal schema, search and geocoding, the routing algorithms, licensing, and a phased roadmap. Safe2Go is much closer than a fresh start: the missing piece is the global connector graph, the intermodal transfer table and a schedule-aware journey engine, not another API.

Key decisions:

1. **One graph, one model.** Train, bus, metro, tram, ferry, coach and flight are all rows in one `mobility_connection` table; walking, taxi, airport and border transfers are rows in one `transfer` table.
2. **GTFS in, GTFS not required.** Mobility Database and national access points are the first acquisition channel, but the internal model also accepts NeTEx, SSIM-style flight schedules and hand-built operator adapters.
3. **Download, store, then route.** Feeds are pulled into a versioned data lake and normalised into PostgreSQL/PostGIS; nothing at query time calls a feed provider.
4. **APIs only for what changes by the minute:** delays, cancellations, vehicle positions, availability, fares, traffic, weather.
5. **Every row knows where it came from:** `source_id`, `confidence` and `last_verified` on every node and connection, so stale or curated data is visible.

Where Safe2Go stands today

Safe2Go already owns most of the raw material for an API-independent engine; what it lacks is the layer that joins it into one graph. Figures below are Postgres estimates from the live inventory of 2026-09-26.

Asset	Size today	Status	What it enables
GTFS database (<code>gtfs</code>)	505 GB	Live	All timetabled transit
<code>stop_times</code>	586.3M rows (266 GB)	Live	Every scheduled stop event
<code>trips</code>	28.7M	Live	Vehicle runs
<code>stops</code>	5.40M	Live	Timetabled stops/stations
<code>routes</code> / <code>agencies</code> / <code>feeds</code>	315K / 8,889 / 2,938	Live	Lines, operators, feed register
<code>route_connections</code>	113.5M (39 GB)	Live	Stop A→B served, trips/day, sample time
<code>realtime_feeds</code>	20	Partial	Live delays for 20 operators only
<code>place_search_index</code>	27.2M	Live	Own forward search
Building footprints (Google + Microsoft)	671.6M	Live	Snap points to real buildings
<code>flight_routes</code> / <code>flight_airlines</code>	67,184 / 6,162	Live, OpenFlights	Airport-pair gate, no schedules
<code>osm_transit</code>	285 GB	Live	OSM stops, addresses, localities, route cache
Valhalla	8 isolated regions	Live	Road routing within one region
<code>pmtiles</code>	13 areas on disk, 7 wired	Partial	Own base map
<code>mobility_graph</code> hours, surcharges, rideshare tables	0 rows	Empty	Falls back to hard-coded defaults

The current planner (`planJourney()`) already does the right thing on trust: it tiers results as live, scheduled, heuristic or on-demand, drops heuristic routes longer than 3x the real road distance, and returns `noReliableRoute` instead of guessing. Keep that discipline in every new layer.

The structural gaps, from the inventory's own missing-data list and the architecture review:

1. **No global graph.** The 8 Valhalla regions (germany, india, usa, msb, australia, canada, africa, europe) do not connect; a cross-region road trip falls to the public OSRM demo server.
2. **Flights check one direct airport pair only.** No hub connections, no schedules, no fares; OpenFlights data is historical.
3. **Transit stops at one change.** `findTransferTrip` handles one transfer; 2+ change journeys never appear as live.
4. **Ferries are a curated hub list** (Genoa, Naples, Piraeus...) with no real sailing times.
5. **No intermodal transfer table:** train→airport→flight timing is buffers in code, not data.
6. **Search and stops still fall back to 9 external services** (Photon, Nominatim, LocationIQ, OpenCage, TomTom, Mappls, Overpass, Geoapify, Foursquare).
7. **shapes.txt not ingested**, so transit leg distance is estimated from road distance.

Three data classes

API independence comes from splitting data by how fast it changes: own the static, refresh the semi-static on a schedule, and call out only for the dynamic.

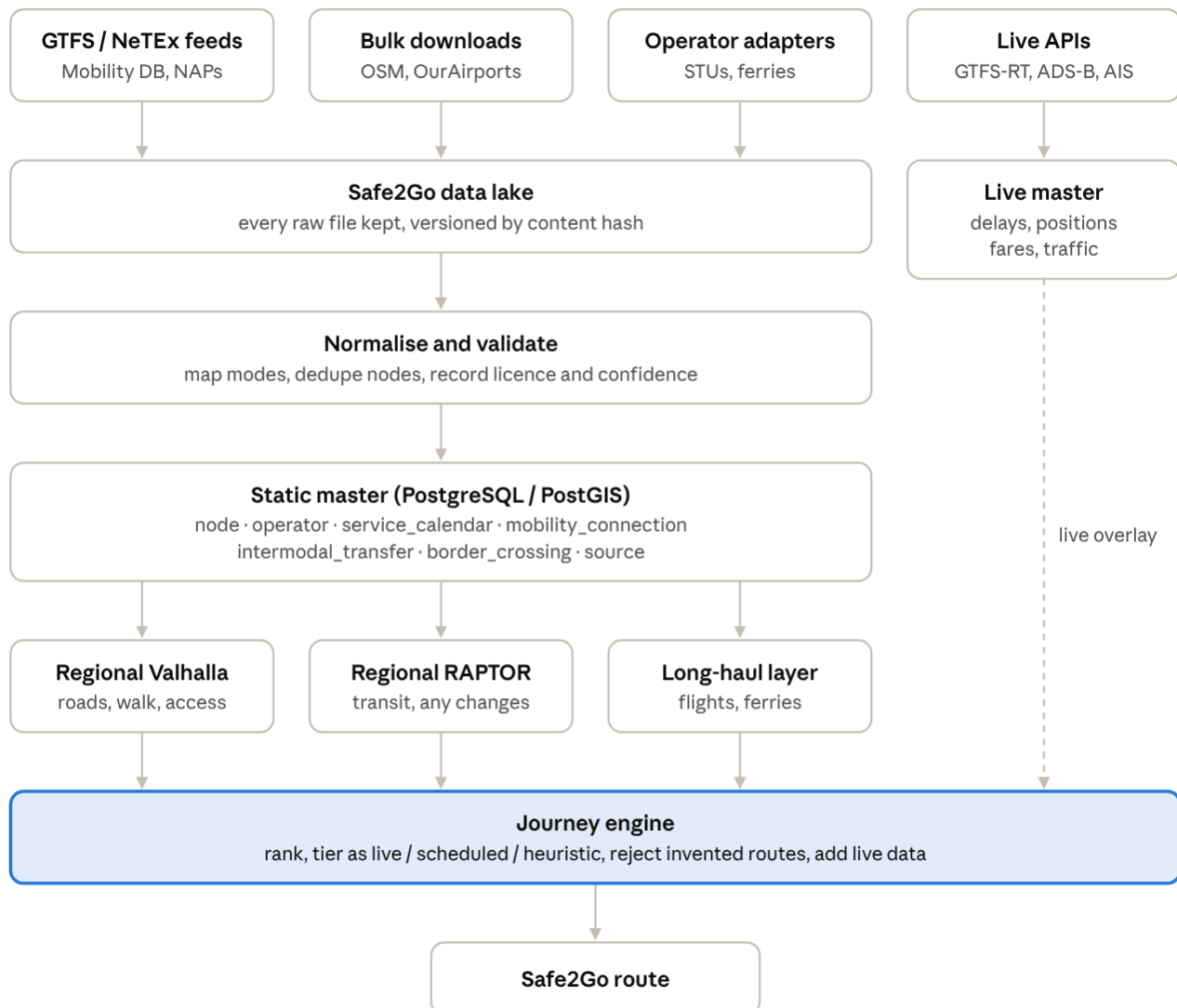
Class	Examples	How Safe2Go gets it	Refresh	Runtime API?
A. Static (own it)	Airports, stations, stops, ports, road network, operators, border crossings, walking links, admin hierarchy	Bulk downloads (OSM, OurAirports, GTFS stops, UNECE/OSM borders)	Monthly or on change	No
B. Semi-static (refresh it)	Train, bus, metro, ferry and flight timetables, operating calendars, seasonal sailings, new routes	GTFS/NeTEx feeds, schedule files, operator adapters	Daily to seasonal, per source	No
C. Dynamic (call for it)	Delays, cancellations, vehicle and aircraft positions, ferry cabin/vehicle space, fares, traffic, closures, weather	GTFS-RT, SIRI, ADS-B, AIS, commercial APIs	Seconds to minutes	Yes, optional

Rule of thumb: if a Class C source disappears, Safe2Go still returns a valid scheduled journey and simply drops the live overlay.

Target architecture

One pipeline feeds one static master; routing reads only from that master, and live APIs join at the very end as an overlay.

Every source lands in one static master; APIs only feed the live overlay



target architecture · 7 layers, live data on a side path

If every box on the dashed path disappeared, Safe2Go would still return a valid scheduled journey. The three engines sit side by side because each handles one kind of edge; the journey engine stitches their results through shared nodes.

Universal data model

Six core tables carry every mode: nodes, operators, services, calendars, connections and transfers, plus a source register that every row points to.

Table	One row is	Key columns
source	One feed, file, adapter or API	source_id, kind (gtfs, netex, ssim, adapter, curated, api), url, license, fetched_at, version_hash
node	Any place you can board, alight or transfer	node_id, node_type (stop, station, platform, airport, port, border, poi), name, geom, country, parent_node_id, external_ids (GTFS stop_id, IATA, ICAO, UN/LOCODE, UIC)
operator	An agency, airline, ferry or bus company	operator_id, name, country, iata_code, website
service_calendar	A pattern of operating dates	service_id, days_mask, valid_from, valid_to, plus an exceptions table for added/removed dates
mobility_connection	One scheduled vehicle hop from A to B	see SQL below
transfer	A non-scheduled link between two nodes	from_node_id, to_node_id, kind (walk, taxi, car, bike, airport_shuttle, check-in, border), duration_s, min_transfer_s, source_id

Station hierarchy matters: "Frankfurt" (city) → "Frankfurt (Main) Hbf" (station) → platform 12. The journey engine searches platforms and stops; users search cities and stations.

```
CREATE TABLE mobility_connection (
  connection_id      BIGSERIAL PRIMARY KEY,
  trip_id            TEXT NOT NULL,          -- one vehicle run
  seq                INT  NOT NULL,          -- hop order within the trip
  from_node_id       BIGINT NOT NULL REFERENCES node,
  to_node_id         BIGINT NOT NULL REFERENCES node,
  mode               TEXT NOT NULL,          -- metro, tram, bus, coach,
```

```

rail, ferry, flight, cable
  route_type          INT,                -- original GTFS / extended
code
  operator_id         BIGINT REFERENCES operator,
  service_id          TEXT NOT NULL REFERENCES service_calendar,
  departure_s         INT NOT NULL,        -- seconds after local
midnight; may exceed 86400
  arrival_s           INT NOT NULL,
  tz                  TEXT NOT NULL,       -- IANA zone of departure node
  arrival_day_offset  SMALLINT DEFAULT 0, -- overnight ferries, long-
haul flights
  min_transfer_s      INT,                -- MCT at the arrival node, if
mode-specific
  border_required     BOOLEAN DEFAULT FALSE,
  bookable            BOOLEAN,
  source_id           BIGINT NOT NULL REFERENCES source,
  confidence          REAL CHECK (confidence BETWEEN 0 AND 1),
  last_verified       TIMESTAMPTZ
);
CREATE INDEX ON mobility_connection (from_node_id, departure_s);
CREATE INDEX ON mobility_connection (trip_id, seq);

```

Two details the original sketch missed: times must be stored with a time zone (a FRA→DEL flight crosses 3.5 hours of offset), and GTFS allows times past 24:00 for trips that run after midnight.

Ground public transit sources

Mobility Database is the first acquisition channel, national access points (NAPs) the second, and city portals plus community feeds fill the rest. Safe2Go already holds 2,938 feeds; the job now is coverage and freshness, not plumbing.

Global catalogues

Catalogue	What it gives	Access	Notes
Mobility Database	6,000+ GTFS, GTFS-RT and GBFS feeds; checks each producer URL daily at midnight UTC and archives every dataset	Free. API needs a refresh token → hourly access token. Keyless full catalogue as CSV: https://files.mobilitydatabase.org/feeds_v2.csv	Replaced TransitFeeds in Feb 2024; TransitFeeds history migrated Dec 2025. Each feed lists a licence URL and bounding box
Transitland	Feed registry, historical archive, GraphQL/REST, Mobility Database ID crosswalk	Free tier with API key; commercial for heavy use	Good second opinion on feed liveness
EU National Access Points	Country-wide GTFS and NeTEx under EU Delegated Regulation 2017/1926	Mostly free, some registration	NeTEx is often richer than the GTFS export

India

Network	Source	Freshness check	Notes
Delhi Metro (DMRC)	OTD Delhi /data/staticDMRC	Last updated Aug 10, 2023: 262 stops, 36 routes, 17,997 trips, 411,686 stop times	Download button opens a usage-purpose form; no published direct URL. Stale: new lines since 2023 missing
Delhi buses (DTC + cluster)	OTD Delhi /data/static	Last updated Jun 20, 2024: 3,464 stops, 543 routes, 16,562 trips, 378,324 stop times	Real-time feed needs a registered key

Network	Source	Freshness check	Notes
Bengaluru BMTC	Mobility Database mdb-2595 (community, Vonger)	Service Jul 2026–Jul 2027, 4,381 routes	Community-built, not official
Indian Railways	mdb-2867 (unofficial)	10,503 routes; dataset window ended Jan 2026	Check for a newer build before relying on it
APSRTC (Andhra Pradesh)	mdb-3050 (community)	3,151 routes; window ended Jan 2026	Same author pattern as Indian Railways feed
Kochi Metro	tld-1031 , producer kochimetro.org/opendata	Service ended Dec 31, 2025	India's first official GTFS (2018); needs a refresh
Kochi buses and boats	mdb-1835	2022–23 window, 473 routes incl. ferries	Historic; useful for stop geometry only
Mumbai BEST, TMT, KDMT, VVMT	Community GTFS projects (e.g. the Mumbai GTFS work listed by FOSS United grantees)	Verify per build	The earlier chat's mdb-3138 download link was not verified
Pune PMPML, Ahmedabad, Rajkot, Hubli-Dharwad BRT	Same community projects	Verify per build	Treat as confidence ≤ 0.7

India rule: official feeds are few and often stale, community feeds are fresher but unofficial. Store both, mark `confidence`, and prefer whichever has a valid service window for the travel date.

Country starting points (outside India)

Country	Where to look	Format
Germany	DELFI nationwide feed via opendata-oeponv.de / Mobilithex	GTFS, NeTEx

Country	Where to look	Format
France	transport.data.gouv.fr	GTFS, NeTEx, GTFS-RT
Switzerland	opentransportdata.swiss	GTFS, NeTEx, GTFS-RT
Netherlands	gtfs.ovapi.nl (OVapi) and NDOV Loket	GTFS, GTFS-RT
Norway	Entur developer portal (all modes incl. ferries)	NeTEx primary, GTFS aggregate
Sweden / Denmark / Finland	Trafiklab / Rejseplanen / Fintraffic	GTFS, NeTEx
UK	Bus Open Data Service (bus-data.dft.gov.uk); National Rail data portal for rail	GTFS, TransXChange, CIF
Ireland	Transport for Ireland transit data	GTFS, GTFS-RT
USA	Per agency, indexed by Mobility Database and the FTA NTD GTFS weblinks	GTFS, GTFS-RT
Japan	ODPT and GTFS-JP regional feeds	GTFS-JP, ODPT API
Singapore	LTA DataMall (routes, stops, arrivals)	Proprietary JSON → adapter
Australia	TfNSW, PTV, Translink, Transperth open data	GTFS, GTFS-RT

URLs and login requirements change; confirm each at ingestion time and record it in `source`.

Mode classification (route_type)

Map every feed's `route_type` to one Safe2Go `mode` at ingestion, using both the basic GTFS codes (0–12) and the extended hierarchical codes (100–1700). Store the original code too.

Safe2Go mode	Basic GTFS	Extended codes
<code>rail</code> (regional, intercity, high-speed, sleeper, car-train)	2	100–117
<code>coach</code> (long-distance bus)	—	200–209

Safe2Go mode	Basic GTFS	Extended codes
metro (subway, underground, urban rail)	1	400–405 (plus 500, 600 in older feeds)
bus (incl. express, night, demand-responsive)	3, 11 (trolleybus)	700–716, 800 (trolleybus), 1500–1507 (taxi/DRT, flag separately)
tram (tram, light rail, cable tram)	0, 5	900–906
ferry (ferries, water buses, water taxis)	4	1000–1021 (1200 = ferry in some feeds)
cable (aerial lift, gondola, funicular)	6, 7	1300–1307, 1400
monorail	12	405
flight	—	1100–1114 (rare in GTFS; flights come from the flight pipeline)

```

MODE_RANGES = [
    ("rail",      {2} | set(range(100, 118))),
    ("coach",     set(range(200, 210))),
    ("metro",     {1, 500, 600} | set(range(400, 405))),
    ("monorail",  {12, 405}),
    ("bus",       {3, 11, 800} | set(range(700, 717))),
    ("drt",       set(range(1500, 1508))),
    ("tram",      {0, 5} | set(range(900, 907))),
    ("ferry",     {4, 1200} | set(range(1000, 1022))),
    ("cable",     {6, 7, 1400} | set(range(1300, 1308))),
    ("flight",    set(range(1100, 1115))),
]

```

```

def to_mode(route_type) -> str:
    try:
        code = int(str(route_type).strip())
    except ValueError:
        return "unknown"
    for mode, codes in MODE_RANGES:
        if code in codes:
            return mode
    return "unknown"

```

Cast to integer after trimming whitespace: some European feeds ship values like "401 ". Log every unknown so new codes are mapped, not silently dropped. Fixes vs the earlier chat: monorail is 405 in the extended set (not part of 400–404 "metro"), and ferry codes

run to 1021, not 1013.

Rail, coach and private bus

Booking platforms (RedBus, MakeMyTrip, ixigo, IntrCity, NueGo, FlixBus) are data sources and booking deep-links, never the routing engine. Where no structured timetable exists, Safe2Go writes an adapter that turns the operator's published schedule into the same `mobility_connection` rows as any GTFS feed.

Adapter pattern

Every non-GTFS source implements one small interface, so adding an operator is a day of work, not a project:

```
class OperatorAdapter:
    source_id: str          # e.g. "in-msrtc-web"
    refresh: str            # "daily" | "weekly" | "seasonal"
    def fetch(self) -> bytes: ...          # raw page / PDF /
    JSON, saved to the lake
    def parse(self, raw) -> list["RawTrip"]: ...      # operator-specific
    def normalise(self, trips) -> "GtfsLikeBundle": ... # stops, trips,
    stop_times, calendar
    def validate(self, bundle) -> "Report": ...      # MobilityData GTFS
    validator rules
```

The adapter outputs a GTFS-like bundle, so the existing ingestion path, validator and `route_connections` builder all apply unchanged.

Priority list

Segment	Examples	Best source	Confidence
Indian state transport (STUs)	MSRTC, UPSRTC, RSRTC, Haryana Roadways, KSRTC, TNSTC, APSRTC, TGSRTC	Official ORS/booking timetables via adapter; community GTFS where it exists	0.6–0.8
Indian Railways	~13,000 daily passenger trains	Community GTFS (mdb-2867) + periodic rebuild from public timetables	0.7
Indian private intercity bus	IntrCity, NueGo, Zingbus, thousands of small operators	Operator sites; aggregator data only under a written licence	0.5–0.7

Segment	Examples	Best source	Confidence
European coach	FlixBus, BlaBlaCar Bus, Alsa, National Express	FlixBus publishes GTFS (listed in Mobility Database); others via NAP feeds	0.8–0.9
European rail	DB, SNCF, Trenitalia, ÖBB, SBB, Renfe	NAP NeTeX/GTFS; DB via DELFI	0.9
North America intercity	Amtrak, VIA Rail, Greyhound	Published GTFS	0.9

Never scrape behind a login or against a site's terms; where a platform is the only source, negotiate a data licence. Section on licensing covers this.

Ferries

Replace the hard-coded ferry hub list (Genoa, Naples, Piraeus...) with a Port Master and a Sailing Master, so a ferry is just another `mobility_connection` with `mode = 'ferry'`. Urban and short-sea ferries already arrive through GTFS; long-distance and overnight ferries need adapters.

Source hierarchy

1. Official GTFS/NeTeX that already includes ferries (Norway's Entur, Washington State Ferries, Sydney Ferries, Kochi Water Metro where published, Hong Kong, Istanbul, Danish and Greek NAP data).
2. Official operator timetable pages or PDFs → operator adapter (GNV, Grimaldi, Tirrenia/Moby, Corsica Ferries, Stena, DFDS, P&O, Brittany Ferries, Tallink Silja, Viking Line, Blue Star, Seajets, Indian coastal and Andaman services).
3. Port authority schedules (validation and gap-fill).
4. National open data.
5. Licensed bulk data from an aggregator (Ferryhopper, FerryGateway/Ferryscanner, Direct Ferries) under contract.
6. Safe2Go-curated entries, marked `confidence ≤ 0.5` and `last_verified`.

Ferry-specific fields

Field	Why
<code>arrival_day_offset</code>	Overnight sailings (Genoa 21:30 → Palermo 07:00 next day)
<code>season_from</code> / <code>season_to</code>	Many Mediterranean routes run Jun-Sep only
<code>vehicles_allowed</code> , <code>foot_passengers_allowed</code>	Car journeys can only use car ferries
<code>check_in_close_min</code>	Becomes the minimum transfer time into the sailing (often 30–120 min, higher with vehicles)
<code>port_node_id</code> per terminal	Large ports have several terminals far apart

Live layer (optional)

AIS gives live ship positions, not schedules. Use it to show "vessel left port 12 min late". Options: AISHub (community exchange, you contribute a receiver), aisstream.io (free websocket), MarineTraffic or VesselFinder (commercial). Fares, cabin and vehicle availability only come from the operator or a licensed aggregator.

Flights

The fastest win in the whole roadmap: one-stop hub search over the 67,184

`flight_routes` Safe2Go already has, so Mannheim → Dar es Salaam stops returning "no flight". The durable fix is a Flight Schedule Master, so connections are checked against real departure times rather than route existence.

Five masters

Master	Content	Source	Licence
Airport	IATA/ICAO, coords, timezone, terminals, city node	OurAirports (public domain) + OSM aerodromes	Free
Airline	IATA/ICAO, alliance, active flag	OpenFlights seed + manual active list	ODbL / curated

Master	Content	Source	Licence
Route	Airport pair, operating airlines	OpenFlights as seed only (frozen since ~2014); refresh from schedules	ODbL
Schedule	Flight number, days, dep/arr local time, validity, equipment	See options below	Varies
Connection rules	Minimum connection time per airport and transfer type	Defaults now; licensed IATA MCT data later	Licensed

Schedule options, cheapest first

1. **Observed schedules (own data, free).** Pull departures and arrivals per airport from OpenSky's `/flights/departure` and `/flights/arrival` for the past 2–4 weeks, group by callsign and weekday, and infer a weekly pattern. Covers most scheduled traffic seen by ADS-B; misses new routes and seasonal changes until observed. Mark confidence 0.6.
2. **Airline-published timetables.** Many carriers publish timetable PDFs or SSIM files; adapt the top 50 airlines that serve Safe2Go's markets.
3. **Aviation Edge** sells periodic database downloads of routes and schedules, which fits the "download, store, route" model better than per-query APIs.
4. **OAG or Cirium** SSIM schedule files: the industry standard, near-complete coverage, enterprise pricing.
5. **FlightAware AeroAPI:** per-query, use only for live status ("QR XXX is delayed 47 min").

Connection logic

A two-leg itinerary is valid when:

```
\text{arr}_{\text{leg1}} + \text{MCT}_{\text{airport, type}} \leq \text{dep}_{\text{leg2}}
```

with all times converted to UTC first. Starting MCT defaults until licensed data arrives: 45 min domestic→domestic, 60 min international at small airports, 90 min at large hubs, 120 min when changing terminals or self-transferring between unallied airlines. Self-transfers must be flagged to the user: a missed connection is their risk.

Ground legs to and from airports use the transfer table: check-in close before departure (domestic 45 min, international 60–90 min) and an arrival buffer for immigration and baggage.

Live layer (optional)

OpenSky gives live ADS-B positions for free. Since **March 18, 2026** it accepts only OAuth2 client credentials; username/password basic auth is gone. Anonymous calls get 400 credits a day, registered users 4,000, active feeders about 8,000, with a larger bounding box costing more credits. Commercial use needs a separate licence from OpenSky.

Transfers, roads and borders

Keep Valhalla as eight-plus regional road engines and put a Global Connector Graph above them; never build one planet-wide Valhalla. The journey engine only sees node → edge → node and does not care which region produced a road leg.

Intermodal transfer table

```
CREATE TABLE intermodal_transfer (
  from_node_id      BIGINT REFERENCES node,
  to_node_id        BIGINT REFERENCES node,
  kind              TEXT,          -- walk, taxi, car, bike, shuttle,
station_to_airport, port_access, border
  distance_m        INT,
  walk_s            INT,
  min_transfer_s    INT,          -- includes check-in, security, gate walk
  max_transfer_m    INT,
  terminal          TEXT,
  security_required BOOLEAN,
  border_required   BOOLEAN,
  source_id         BIGINT REFERENCES source,
  confidence        REAL,
  PRIMARY KEY (from_node_id, to_node_id, kind)
);
```

How to fill it, in order:

1. **GTFS transfers.txt** and **pathways.txt** from every feed (part of the existing ~2.9M official transfers).
2. **Walk links** between any two stops within 400 m (800 m at stations and airports), timed with Valhalla pedestrian costing, not straight-line distance.

3. **Station ↔ airport and station ↔ port links** for every airport and port node: walking where they share a building (FRA Fernbahnhof), otherwise the fastest scheduled or taxi leg.
4. **Mode rules as data:** airport check-in close, ferry check-in close, security time, all from the `mobility_graph` tables that are empty today (`mode_service_hours` , `place_fare_surcharge`).

Global connector graph

Every node that matters across regions (airports, ports, major stations, border crossings) gets a precomputed road leg to its nearest gateway nodes, computed by the Valhalla region it sits in. Cross-region trips then chain: Valhalla Germany → border node → Valhalla Europe, or road → FRA → flight → SIN → MRT.

Border Crossing Master

Column	Example
<code>border_id</code> , <code>country_a</code> , <code>country_b</code>	DE-AT, Kiefersfelden/Kufstein
<code>lat</code> , <code>lon</code> , <code>road_osm_way_ids</code>	From OSM <code>barrier=border_control</code> and border ways
<code>vehicle_types</code>	car, truck, bus, pedestrian
<code>opening_hours</code>	24/7 or schedule
<code>restrictions</code>	Visa, permit, closed to foreigners

Seed it from OSM border-control nodes and the UNECE border-crossing GIS layer, then curate high-traffic crossings. Valhalla regions should overlap by about 20–50 km at their edges so border nodes exist in both graphs.

Remove the OSRM public server

It is the routing fallback outside the 8 regions and has no SLA. Close the gap in this order: wire the 6 tile sets already on disk (argentina, brazil, china, europe, mexico, russia) into `regionBounds.js` ; build Valhalla for Latin America, Middle East, East Asia, rest of Southeast Asia and New Zealand from Geofabrik extracts; then return `connectorUnavailable` instead of calling OSRM.

Search, geocoding and reverse geocoding

Search should answer from Safe2Go's own 27.2M-row index in under 100 ms, with external geocoders reduced to a logged, rate-limited fallback and then switched off country by country.

Forward search pipeline

1. **Normalise:** lowercase, strip diacritics, transliterate (Devanagari → Latin and back), expand abbreviations ("str" → "strasse", "sec" → "sector"), drop house-number noise.
2. **Exact and alias match:** `locality_alias` (user-confirmed) and official names.
3. **Prefix (as-you-type):** "sa" → Sahibabad, Saharanpur, Saket, Salem; "satyam e" → Satyam Enclave. Edge n-gram index, ranked by population, importance and distance from the user.
4. **Token and fuzzy:** `pg_trgm` similarity for typos.
5. **Spatial bias:** boost results near the user or the map view.
6. **Snap:** optional nearest building from the 671.6M footprints.

```
CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE INDEX psi_prefix ON place_search_index (name_norm text_pattern_ops);
CREATE INDEX psi_trgm ON place_search_index USING gin (name_norm
gin_trgm_ops);
-- as-you-type
SELECT name, lat, lon FROM place_search_index
WHERE name_norm LIKE 'satyam e%'
ORDER BY importance DESC LIMIT 10;
```

At 27M rows and rising, a dedicated engine (Typesense, Meilisearch or OpenSearch with edge n-grams) will hold sub-100 ms more easily than Postgres; keep Postgres as the source of truth.

Reverse geocoding (GPS → address)

Return what the evidence supports, with a confidence, rather than inventing a plot number:

```
GPS → nearest road → nearest locality → nearest building → address evidence →
confidence
```

```
Rajendra Nagar, Ghaziabad, Uttar Pradesh 201005 · confidence 0.62
```

Add DIGIPIN (computed locally) for India and a house number only when an addressed building or interpolation range actually covers the point.

External services to retire

Service	Replace with	When
Overpass (3 mirrors)	Local OSM in PostGIS via osm2pgsql	Now: OSM is already downloaded
Geoapify, Foursquare	<code>transit_stops</code> + GTFS stops + OSM POIs	Now
ipwho.is	Local IP database (DB-IP Lite or MaxMind GeoLite2)	Now
Photon, LocationIQ, OpenCage, TomTom Search	<code>place_search_index</code> per country	As each country import finishes
Nominatim reverse	<code>geo_address</code> + OSM addresses + buildings	After nearest-address query is built
Mappls	<code>locality_alias</code> + India Post + LGD + India OSM	Last: grows with real usage

Journey engine

Use a two-level search: RAPTOR inside each region for local transit, and a sparse global layer of long-haul connections (flights, overnight ferries, long-distance rail and coach) between gateway nodes. Extending `findTransferTrip` with more SQL joins will not scale to 586M stop times.

Algorithm choices

Algorithm	Best for	Why it fits Safe2Go
RAPTOR / McRAPTOR	Regional public transport, any number of changes	Round-based (round $k = k$ transfers), no preprocessing, multi-criteria (time, transfers, cost)
Connection Scan (CSA)	Earliest arrival over a time-sorted connection array	Maps directly onto <code>mobility_connection</code> ; very simple and cache-friendly
Trip-Based routing / Transfer Patterns	Very large networks, fast repeated queries	Precomputation pays off once feeds stabilise

Algorithm	Best for	Why it fits Safe2Go
Time-dependent Dijkstra / A*	Sparse long-haul layer (flights, ferries)	Few nodes ($\approx 4,000$ airports, a few thousand ports)
Valhalla	Road, walk, bike legs and access/egress	Already deployed

Query flow

1. Geocode origin and destination from the own index.
2. **Access/egress:** Valhalla isochrone or nearest-stop search, giving candidate stops within walk, bike, taxi or car range.
3. **Local RAPTOR** in the origin region to reach both the destination (if same region) and gateway nodes (airports, ports, intercity stations).
4. **Long-haul layer:** time-dependent search from reached gateways, honouring MCT and check-in rules from `intermodal_transfer`.
5. **Local RAPTOR** in the destination region from arrival gateways.
6. **Pareto filter** on arrival time, number of changes, cost and walking; tag each leg live, scheduled, heuristic or on-demand.
7. **Validate:** keep `rejectFabricatedRoutes()` and `noReliableRoute`.
8. **Live overlay:** GTFS-RT delays, flight status, AIS, traffic.

Build or adopt

Two open-source engines already implement this on GTFS at continental scale and are worth benchmarking before writing a new one: **MOTIS** (powers the community planner Transitous, which ingests feeds worldwide) and **OpenTripPlanner 2** (RAPTOR-based, NeTEx and GTFS). Either can run per region as the local layer while Safe2Go owns the long-haul layer, the node model and the ranking.

Performance notes

- Load connections into memory as flat arrays sorted by departure, per region and per service day; keep Postgres for storage, not for search.
- Expand calendars to concrete service days for a rolling 7–14 day window.
- Store times as UTC seconds for search and convert at output; overnight trips (24:00+) and date-line flights are the usual bugs.
- Keep `route_connections` (113.5M rows) as a fast "is there any service" prefilter and for the scheduled tier.

Ingestion pipeline

The pipeline below replaces the earlier chat's script, which would not have run: the Mobility Database API needs a token, whole feeds were held in memory, and every country was appended into one SQLite file. This version reads the keyless catalogue CSV, stores every zip in the lake by content hash, skips unchanged feeds, validates, and bulk-loads into Postgres staging schemas.

```
"""safe2go_gtfs_sync.py – discover, download, version and stage GTFS
feeds."""
import csv, hashlib, io, os, pathlib, subprocess, zipfile
import requests, psycopg

CATALOG = "https://files.mobilitydatabase.org/feeds_v2.csv"
LAKE = pathlib.Path(os.getenv("S2G_LAKE", "/data/lake/gtfs"))
PG = os.getenv("S2G_PG", "postgres://safe2go@localhost/gtfs")
COUNTRIES = set(filter(None, os.getenv("S2G_COUNTRIES", "").split(","))) #
empty = all
TABLES = ["agency", "stops", "routes", "trips", "stop_times", "calendar",
          "calendar_dates", "transfers", "pathways", "shapes", "frequencies",
          "feed_info"]
UA = {"User-Agent": "Safe2Go-Ingest/2.0 (contact: data@safe2go.example)"}

def catalogue():
    text = requests.get(CATALOG, headers=UA, timeout=60).text
    for row in csv.DictReader(io.StringIO(text)):
        if row.get("data_type") != "gtfs" or row.get("status") in
("deprecated", "inactive"):
            continue
        cc = row.get("location.country_code", "")
        if COUNTRIES and cc not in COUNTRIES:
            continue
        url = row.get("urls.latest") or row.get("urls.direct_download") #
hosted copy first
        if url:
            yield row["id"], cc, row.get("provider", ""), url,
row.get("urls.license", "")

def download(feed_id, url):
    tmp = LAKE / "_tmp" / f"{feed_id}.zip"
    tmp.parent.mkdir(parents=True, exist_ok=True)
    h = hashlib.sha256()
    with requests.get(url, headers=UA, timeout=300, stream=True) as r:
        r.raise_for_status()
        with open(tmp, "wb") as f:
            for chunk in r.iter_content(1 << 20):
```

```

        f.write(chunk); h.update(chunk)
    digest = h.hexdigest()
    final = LAKE / feed_id / f"{digest[:16]}.zip"
    if final.exists():
        tmp.unlink(); return final, digest, False        # unchanged
    final.parent.mkdir(parents=True, exist_ok=True)
    tmp.rename(final)
    return final, digest, True

def validate(path):
    # MobilityData's canonical validator (Java); returns non-zero on fatal
    errors
    out = path.with_suffix(".report")
    return subprocess.run(["java", "-jar", "gtfs-validator-cli.jar",
                           "-i", str(path), "-o", str(out)]).returncode == 0

def stage(conn, feed_id, path):
    schema = f"stg_{feed_id.replace('-', '_)}"
    with conn.cursor() as cur, zipfile.ZipFile(path) as z:
        cur.execute(f"DROP SCHEMA IF EXISTS {schema} CASCADE; CREATE SCHEMA
{schema}")
        for t in TABLES:
            name = next((n for n in z.namelist() if n.endswith(f"{t}.txt")),
None)
            if not name:
                continue
            with z.open(name) as fh:
                header = fh.readline().decode("utf-8-sig").strip().split(",")
                cols = ", ".join(f'"{c.strip()}" text' for c in header)
                cur.execute(f"CREATE TABLE {schema}.{t} ({cols}")
                with cur.copy(f"COPY {schema}.{t} FROM STDIN WITH (FORMAT
csv)") as cp:
                    while chunk := fh.read(1 << 20):
                        cp.write(chunk)

    conn.commit()
    return schema

def main():
    with psycopg.connect(PG) as conn:
        for feed_id, cc, provider, url, licence in catalogue():
            try:
                path, digest, changed = download(feed_id, url)
                if not changed:
                    continue
                if not validate(path):
                    print(f"[warn] {feed_id} failed validation; staged with

```

```

low confidence")
    schema = stage(conn, feed_id, path)
    with conn.cursor() as cur:
        cur.execute("""INSERT INTO source (source_id, kind, url,
license, version_hash, fetched_at, country)
                        VALUES (%s, 'gtfs', %s, %s, %s, now(), %s)
                        ON CONFLICT (source_id) DO UPDATE SET
version_hash=EXCLUDED.version_hash,
                        fetched_at=now(), url=EXCLUDED.url,
license=EXCLUDED.license""",
                        (feed_id, url, licence, digest, cc))
    conn.commit()
    print(f"[ok] {cc} {provider} → {schema}")
except Exception as e:
    conn.rollback()
    print(f"[skip] {feed_id}: {e}")

if __name__ == "__main__":
    main()

```

After staging, a SQL merge step maps each staged feed into the canonical tables: prefix IDs with the feed id, map `route_type` → `mode`, dedupe stops against existing nodes (same name within 50 m), and rebuild `mobility_connection` and `route_connections` for that feed only.

For the API instead of the CSV: store the refresh token from your Mobility Database account, exchange it at `POST https://api.mobilitydatabase.org/v1/tokens` for an access token valid one hour, and send it as `Authorization: Bearer ...` to `/v1/gtfs_feeds`. Column names in the CSV can change; check the header once per run and fail loudly.

Live layer: aircraft near an airport

The earlier scripts used username/password auth, which OpenSky stopped accepting on March 18, 2026. This version uses OAuth2 client credentials, caches the token, falls back to anonymous access, and draws the result with Folium.

```

"""live_flights.py – aircraft near an airport from OpenSky, drawn on a Folium
map."""
import os, time
import folium, pandas as pd, requests

TOKEN_URL = "https://auth.opensky-network.org/auth/realms/opensky-
network/protocol/openid-connect/token"
API = "https://opensky-network.org/api/states/all"

```

```

COLS = ["icao24", "callsign", "origin_country", "time_position",
        "last_contact",
        "longitude", "latitude", "baro_altitude", "on_ground", "velocity",
        "true_track", "vertical_rate", "sensors", "geo_altitude", "squawk",
        "spi", "position_source"]
_token = {"value": None, "exp": 0}

def bearer():
    cid, secret = os.getenv("OPENSKY_CLIENT_ID"),
os.getenv("OPENSKY_CLIENT_SECRET")
    if not (cid and secret):
        return {} # anonymous: 400
credits/day
    if time.time() > _token["exp"] - 60:
        r = requests.post(TOKEN_URL, data={"grant_type":
"client_credentials",
                                "client_id": cid, "client_secret": secret},
timeout=15)
        r.raise_for_status()
        j = r.json()
        _token.update(value=j["access_token"], exp=time.time() +
j.get("expires_in", 1800))
        return {"Authorization": f"Bearer {_token['value']}"}

def aircraft_near(lat, lon, half_deg=0.5) -> pd.DataFrame:
    box = {"lamin": lat - half_deg, "lamax": lat + half_deg,
          "lomin": lon - half_deg, "lomax": lon + half_deg} # small box =
fewer credits
    r = requests.get(API, params=box, headers=bearer(), timeout=20)
    if r.status_code == 429:
        raise RuntimeError("OpenSky credits exhausted; poll less often or
authenticate")
    r.raise_for_status()
    df = pd.DataFrame(r.json().get("states") or [], columns=COLS)
    if df.empty:
        return df
    df["callsign"] = df["callsign"].fillna("").str.strip()
    return df.dropna(subset=["latitude", "longitude"])

def draw(df, lat, lon, name, out="live_flights.html"):
    m = folium.Map([lat, lon], zoom_start=10, tiles="CartoDB dark_matter")
    folium.Marker([lat, lon], tooltip=name,
                  icon=folium.Icon(color="red", icon="plane",
prefix="fa")).add_to(m)
    for _, a in df.iterrows():
        label = a.callsign or a.icao24

```

```

        alt = f"{a.baro_altitude:.0f} m" if pd.notnull(a.baro_altitude) else
        "n/a"
        spd = f"{a.velocity * 3.6:.0f} km/h" if pd.notnull(a.velocity) else
        "n/a"
        hdg = f"{a.true_track:.0f}°" if pd.notnull(a.true_track) else "n/a"
        colour = "gray" if a.on_ground else "dodgerblue"
        folium.CircleMarker([a.latitude, a.longitude], radius=6 if
a.on_ground else 8,
                            color=colour, fill=True, fill_opacity=0.8, tooltip=label,
                            popup=f"<b>{label}</b><br>{a.origin_country}<br>Alt {alt}"
        <br>Speed {spd}<br>Heading {hdg}"
        ).add_to(m)
        m.save(out)
        return out

if __name__ == "__main__":
    FRA = (50.0379, 8.5622)
    df = aircraft_near(*FRA)
    print(f"{len(df)} aircraft near FRA")
    print(draw(df, *FRA, "Frankfurt Airport (FRA)"))

```

Run with `pip install requests pandas folium`; set `OPENSKY_CLIENT_ID` and `OPENSKY_CLIENT_SECRET` from an API client created on your OpenSky account page. In Safe2Go this belongs in the Class C live overlay only: it shows where a flight is, not whether a route exists.

Licensing and data quality

Owning data means owning its licence obligations: record the licence of every source in `source.license` and check it before a dataset reaches production. This is not legal advice; have counsel review the commercial-use cases below.

Source	Licence (typical)	What it requires
OpenStreetMap (roads, addresses, POIs, borders)	ODbL 1.0	Attribution; share-alike if you publish a derived database
OurAirports	Public domain	Nothing, attribution appreciated
OpenFlights routes/airlines	ODbL	Attribution, share-alike on the database
GeoNames	CC BY 4.0	Attribution

Source	Licence (typical)	What it requires
Google Open Buildings	CC BY 4.0 / ODbL (per release)	Attribution
Microsoft Building Footprints	ODbL	Attribution, share-alike
GTFS feeds	Varies per feed (CC BY, ODbL, custom terms, some non-commercial)	Read each feed's licence URL from the catalogue
OTD Delhi	Portal terms; asks commercial vs non-commercial at download	Confirm commercial terms with the portal
OpenSky	Free for research/non-commercial	Commercial licence needed
Booking platforms, aggregators	Proprietary	Written data licence; no scraping against terms

Quality gates on every load

- MobilityData GTFS validator: block on fatal errors, store the report.
- Service window covers today + 7 days, otherwise mark the feed stale and lower confidence .
- Stop coordinates inside the feed's country polygon and not at 0,0.
- Speeds between consecutive stops plausible per mode (bus < 120 km/h, rail < 350, flight < 1,000).
- Row-count diff vs the previous version: a drop over 30% needs human review.
- Sample journeys: a fixed regression set (Mannheim → Frankfurt, Sahibabad → Connaught Place, FRA → DEL, Genoa → Palermo) re-run after each load.

Gaps and roadmap

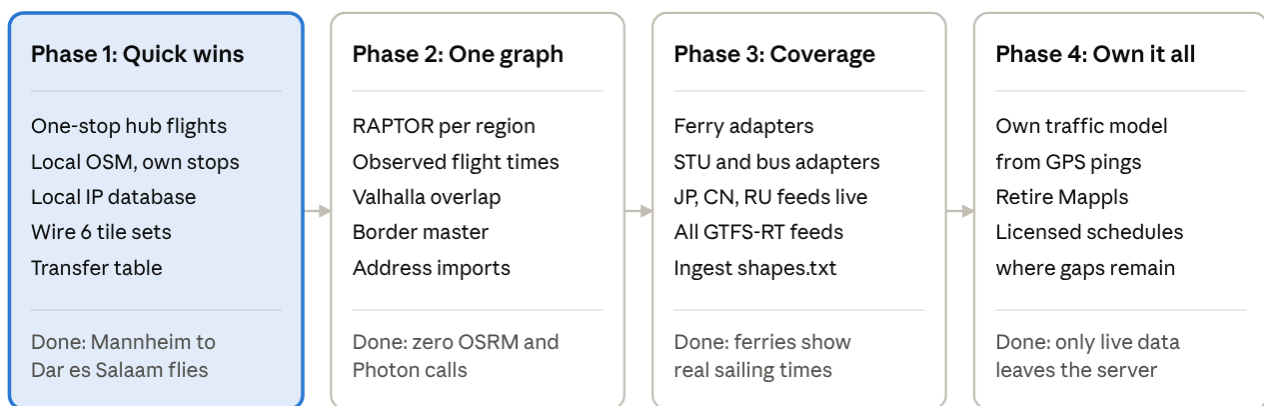
Every routing gap can be closed without a runtime API; only live data (delays, availability, fares, traffic) keeps an external source.

Area	Today	Fix	Runtime API after?	Phase
Connecting flights	One direct airport pair	Hub search over <code>flight_routes</code> , then schedule master	No	1
Overpass, Geoapify, Foursquare, ipwho.is	External fallbacks	Local OSM/PostGIS, own stops, local IP DB	No	1
Map tiles	7 of 13 wired	Wire argentina, brazil, china, europe, mexico, russia	No	1
Intermodal transfers	Buffers in code	<code>intermodal_transfer</code> table, fill <code>mobility_graph</code> rules	No	1
Multi-change transit	Max 1 change	RAPTOR per region (or MOTIS/OTP)	No	2
Flight schedules	None	Observed schedules from ADS-B, then licensed SSIM	No (live status optional)	2
Cross-region road	OSRM public, no SLA	Overlapping Valhalla regions + border master + connector graph	No	2
Address search outside imported countries	Photon, Nominatim, etc.	Per-country imports, street-name pass	No	2
Ferries	Curated hubs, no times	Port/Sailing masters + operator adapters	No (availability optional)	3
Private and state buses (India)	Partial	STU and operator adapters, licensed aggregator data	No	3
Transit coverage (JP, CN, RU set up, not launched)	Uneven	Launch feeds, add NAP NeTEx	No	3
Live delays	20 GTFS-RT feeds	Add every GTFS-RT in the catalogue	Yes, operator feeds	3

Area	Today	Fix	Runtime API after?	Phase
shapes.txt	Not ingested	Ingest for leg geometry and distance	No	3
Traffic	TomTom / Google	Own model from Safe2Go GPS pings + historic speeds	Yes until then	4
Mappls (India hyperlocal)	External	Grows with locality_alias usage	No, eventually	4

Keep permanently: Stripe, Apple IAP, Google sign-in, email, the AI assistant, operator GTFS-RT feeds, and booking deep-links. They are not routing dependencies.

Four phases take Safe2Go from quick wins to owning every route



roadmap · 4 phases, each with its done test

Phase 1 needs no new data, only code over tables Safe2Go already has; each later phase starts when the previous done test passes.

Corrections to the earlier chats

Earlier claim	Status	Correct version
Direct download URLs <code>otd.delhi.gov.in/data/staticDMRC/download/</code> and <code>/data/static/download/</code>	Not published	The portal serves the zip after a usage-purpose form; script it only with the portal's consent

Earlier claim	Status	Correct version
Delhi data is current	Stale	Metro last updated Aug 10, 2023; buses Jun 20, 2024
<code>api.mobilitydatabase.org/v1/gtfs_feeds</code> works without auth	Wrong	Needs a bearer token from a refresh token; the keyless option is the catalogue CSV
"1,000+ agencies" worldwide	Understated	Mobility Database lists 6,000+ feeds; Safe2Go already holds 2,938
OpenSky with <code>auth=(user, password)</code>	Wrong since Mar 18, 2026	OAuth2 client credentials only
Monorail inside metro codes 400–405; ferry codes 1000–1013	Imprecise	Monorail 405 / basic 12; ferry 1000–1021 and 1200
One SQLite file for the world	Will not scale	Postgres/PostGIS for storage, in-memory arrays for search, DuckDB for analytics
Mumbai BEST at <code>mdb-3138</code>	Unverified	Check the Mobility Database entry before use
HMRL (Hyderabad) and Goa GTFS portals	Unverified	Not confirmed; search the catalogue first
OpenFlights routes as the flight network	Partial	Historical seed only; schedules must come from elsewhere

Sources

- [OTD Delhi bus static data](#) and [DMRC static data](#)
- [Mobility Database FAQ](#) and [catalogue repository](#)
- [Mobility Database API client \(auth flow\)](#)

- Feed pages: [BMTC mdb-2595](#), [Indian Railways mdb-2867](#), [APSRTC mdb-3050](#), [Kochi Metro tld-1031](#), [Kochi buses and boats mdb-1835](#)
- [OpenSky REST API docs](#) and [OpenSky API repository](#)
- [Open Transit Data in India \(New America\)](#)
- Safe2Go internal: "Safe2Go — Logic, Databases & APIs" (inventory 2026-09-26) and the architecture review chats supplied with this document