Smart Carpool & Backhaul Matcher — Design Doc

Owner: Guillaume / Storia Eng

Last updated: 2025-08-27

---

1) Problem Summary

We need a single platform that: - Matches **commuter carpools** (drivers ↔ riders). - Matches **logistics backhauls** (shippers' loads ↔ truck drivers' available capacity). - Provides **real-time chat**, **notifications**, and **simple, safe transactions**.

Target outcome: reduce empty seats and empty miles, cut emissions and cost, and make coordination trivial.

---

2) Goals & Non-Goals

Goals - Accurate, low-latency matches based on **time windows, geospatial proximity, and capacity**. - Unified UX for carpool + backhaul with **shared matching engine** and chat. - Mobile-friendly React app with live updates (WebSockets / Socket.IO). - Auditable data with clear status lifecycles and soft-delete/TTL for stale items.

Non-Goals (v1) - Payments/escrow and KYC (can be phased to v2). - Complex route optimization across multiple carriers (keep to single-leg match v1). - Multi-tenant white-label (assume single tenant v1, isolate by org in v2).

---

3) Personas & Core Use Cases

Personas - Carpool Rider: finds a ride daily/occasionally; needs ETA, seat availability, trust. - Carpool Driver: posts ride, sets time, confirms riders, coordinates pickup. - Shipper: posts load (origin/dest, weight, window), needs coverage in a time window. - Truck Driver / Fleet: publishes availability/capacity along route, wants profitable backhauls.

Use Cases 1. Rider searches, requests seat → driver accepts → chat → ride completes. 2. Shipper posts load → matching engine suggests trucks → confirm → chat → delivery. 3. Driver/Truck marks status (open → matched → completed/cancelled), automatic expiry.

---

4) High-Level Architecture

```
[ React (Web/Mobile) ]
   |  REST + WebSocket (Socket.IO)
[ Node.js + Express API ]
   |-- Auth (JWT)
   |-- Carpool / Backhaul APIs
   |-- Matching Service (geospatial + scoring)
   |-- Chat Service (Socket.IO, message persistence)
   |-- Notification Worker (email/SMS/push)
   |-- Scheduler (TTL cleanup, reminders, re-matching)
   |
   |     [ Redis / BullMQ (jobs, rate-limit, sessions optional) ]
   |
[ MongoDB Atlas ]  (2dsphere indexes, schema validation)
```

Infra: Dockerized services, CI/CD; dev/stage/prod. Horizontal scale API and Socket pods; MongoDB Atlas (or self-hosted) with replica set.

---

5) Data Model (MongoDB)

> Collections mirror your proposal with validation + indexes.

5.1 Users

```
{
  "_id": {"$oid": "..."},
  "name": "Isabel",
  "email": "isabel@company.com",
  "passwordHash": "bcrypt$...",
  "role": "carpool_driver|carpool_rider|truck_driver|shipper",
  "phone": "+1-234-567-8901",
  "profilePic": "https://...",
  "department": "Engineering",
  "rating": 4.7,
  "createdAt": {"$date": "2025-08-27T12:00:00Z"}
}
```

Indexes & validation - Unique: `email`. - Sparse: `phone`. - Validation: email format, rating range [0,5].

5.2 CarpoolRides

```
{
  "_id": {"$oid": "..."},
```

```
  "driverId": {"$oid": "userId"},
  "origin": {"type": "Point", "coordinates": [-73.5673, 45.5017]},
  "destination": {"type": "Point", "coordinates": [-73.45, 45.52]},
  "date": {"$date": "2025-08-28T08:00:00Z"},
  "seatsAvailable": 2,
  "ridersConfirmed": [{"$oid":"userId1"},{"$oid":"userId2"}],
  "status": "open|confirmed|completed|cancelled",
  "createdAt": {"$date": "2025-08-27T12:30:00Z"}
}
```

Indexes - `origin` 2dsphere, `destination` 2dsphere. - Compound: `{status:1, date:1}` for searches. - TTL (optional): auto-expire past `date` + grace (e.g., 24h) via scheduler.

5.3 CarpoolRequests

```
{ "rideId": ObjectId, "riderId": ObjectId, "status": "pending|accepted|
rejected", "createdAt": ISODate }
```

Indexes: `{rideId:1, riderId:1}` unique to prevent duplicates; `{status:1}`.

5.4 Loads (Backhaul)

```
{
  "_id": ObjectId,
  "shipperId": ObjectId,
  "origin": {"type":"Point","coordinates":[-73.5673,45.5017]},
  "destination": {"type":"Point","coordinates":[-74.0059,40.7128]},
  "weight": 1200,
  "description": "Pallet of office supplies",
  "deliveryWindow": {"start": ISODate, "end": ISODate},
  "status": "open|matched|completed",
  "createdAt": ISODate
}
```

Indexes: 2dsphere on `origin`, `destination`; compound `{status:1, "deliveryWindow.start": 1}`.

5.5 TruckAvailability

```
{
  "_id": ObjectId,
  "truckDriverId": ObjectId,
  "route": [ {"type":"Point","coordinates":[-73.56,45.50]},
```

```
{"type":"Point","coordinates":[-74.00,40.71]} ],
  "capacity": 2000,
  "availableFrom": ISODate,
  "availableUntil": ISODate,
  "status": "open|matched|completed",
  "createdAt": ISODate
}
```

Indexes - `routePoints` (derived) 2dsphere for corridor search; or store as `LineString` and use `$geoIntersects`. - Compound: `{status:1, availableFrom:1, availableUntil:1}`.

5.6 Matches (carpool + backhaul)

```
{
  "_id": ObjectId,
  "type": "carpool|backhaul",
  "rideOrLoadId": ObjectId,
  "driverOrTruckId": ObjectId,
  "ridersOrShippers": [ObjectId],
  "status": "pending|confirmed|completed|cancelled",
  "score": 0.0,
  "createdAt": ISODate, "updatedAt": ISODate
}
```

Indexes: `{type:1, status:1}`, `{driverOrTruckId:1}`, `{rideOrLoadId:1}`.

5.7 Messages (Chat)

```
{ "_id": ObjectId, "matchId": ObjectId, "senderId": ObjectId, "message":
"...", "timestamp": ISODate }
```

Indexes: `{matchId:1, timestamp:1}`; consider TTL (e.g., 90 days) if desired.

---

6) API (v1)

REST over JSON; JWT bearer auth. Minimal examples shown.

Auth

- `POST /auth/signup` → {token}
- `POST /auth/login` → {token}

Users

- `GET /users/me`
- `PATCH /users/me` (name, phone, department, profilePic)

Carpool

- `POST /rides` (create ride)
- `GET /rides?near=[lon,lat]&dest=[lon,lat]&date=ISO&radiusKm=...` (search)
- `GET /rides/:id`
- `PATCH /rides/:id` (status | seatsAvailable)
- `POST /rides/:id/requests` (rider requests)
- `PATCH /requests/:id` (accept/reject)

Backhaul

- `POST /loads`
- `GET /loads?near=[lon,lat]&dest=[lon,lat]&windowStart=...&windowEnd=...`
- `POST /trucks/availability`
- `GET /trucks/availability?routeNear=[lon,lat]&time=...`

Matches

- `GET /matches/:id`
- `POST /matches/:id/confirm` (双方确认)
- `PATCH /matches/:id` (status)

Chat

- WS: `/ws` (Socket.IO) → join room by `matchId`
- HTTP fallback: `GET /matches/:id/messages` (paged), `POST /matches/:id/messages`

Common: 429 rate-limit, 401/403 auth, 422 validation, 409 conflict (e.g., seat full), idempotency via `Idempotency-Key` header for creates.

---

7) Matching Engine

Pipeline (both domains) 1) **Pre-filter** - Time: overlap(`ride.date` ≈ rider desired time) OR overlap(`deliveryWindow` with `availableFrom/Until`). - Capacity: `seatsAvailable > 0` or `load.weight ≤ truck.capacity`. - Geography: - Carpool: `$geoNear` from rider origin within `R1` km AND ride destination within `R2` km. - Backhaul: load origin/dest within buffers of truck route corridor. 2) **Score** (weighted sum examples) - `Δdistance` detour vs direct route (minimize). - `Δtime` vs preferred time (minimize). - Driver/truck **rating** (maximize). - Historic acceptance rate (maximize). - For backhaul: **capacity utilization** (maximize). 3) **Select top-K** candidates, create `Matches` with `status=pending`, notify parties.

Geospatial details - Carpool: query rides with `$geoNear` on `origin` (e.g., 5–10 km) + filter destination `$near` maxDistance. - Backhaul corridor: store `LineString` for route. - Option A: Pre-compute a **buffer polygon** (100–500 m) and use `$geoWithin`. (Buffer computed by a worker using turf.js or similar.) - Option B: Sample `route` points, run `$near` against each with small `maxDistance` and merge.

Pseudo

```
// carpool example
$geoNear({
  near: { type: 'Point', coordinates: rider.origin },
  key: 'origin', maxDistance: km(R1)
})
.match({ status: 'open', date: { $gte: start, $lte: end } })
.addFields({ destDist: { $function: ['haversine', '$destination',
rider.destination] } })
.match({ destDist: { $lte: km(R2) } })
.addFields({ score: scoringExpression })
.sort({ score: -1 }).limit(K)
```

---

8) Security, Privacy, Compliance

- **Auth**: JWT (RS256). Refresh tokens (httpOnly cookie). Passwords: bcrypt.
- **RBAC**: enforce role per endpoint (rider vs driver vs shipper vs truck).
- **PII**: encrypt at rest (MongoDB Atlas), redact logs. Mask phone/email in public listings.
- **Rate limiting**: per-IP + per-user (e.g., 100 req/min), login brute-force guard.
- **Data retention**: archive or TTL stale rides/loads/messages. GDPR/CCPA delete on request.

---

9) Observability & Ops

- **Logging**: structured JSON (pino/winston). Correlate request-id.
- **Metrics**: Prometheus counters for searches, matches, confirmations, chat msgs; p95 latencies.
- **Tracing**: OpenTelemetry spans for search → match → confirm.
- **Dashboards**: Grafana panels (search volume, conversion, failures, geo heatmap).
- **Alerts**: match failure spikes; queue backlog; Mongo replication lag.

---

10) Performance Targets (SLOs)

- Search API p95 < 300 ms (warm cache) for metro areas, < 800 ms worst.
- Match suggestion availability < 2 s from post.
- Chat message delivery < 200 ms p95.

Scale notes - Add **Redis** caches for hot geo queries. - Use **read replicas**; keep write paths thin. - Consider sharding by region (e.g., country → cityCode) in v2.

---

11) Failure Modes & Mitigations

- **Over-matching / stale results** → verify status & capacity in transaction before confirm; recheck seats/capacity.
- **Geo precision drift** → normalize to 6–7 decimal places; always store lon/lat order.
- **Time zone errors** → store ISO UTC; convert at edges.
- **Chat outages** → auto-reconnect, fall back to HTTP.

---

12) Testing Strategy

- **Unit**: scoring functions, validators, auth.
- **Integration**: `$geoNear` pipelines (use seeded fixtures), request → accept flow.
- **E2E**: Cypress/Playwright for rider ↔ driver flows; shipper ↔ truck flows.
- **Load tests**: k6/Gatling for search and chat fan-out.

---

13) Release Plan

MVP (4–6 weeks) - Auth, Users - Post/search CarpoolRides + Requests; confirm flow - Post/search Loads + TruckAvailability; confirm flow - Basic Matching (pre-filter + simple scoring) - Chat on confirmed matches - Basic metrics + logs

v1.1 - Corridor buffers, richer scoring - Notifications (email/SMS) - Reminders & expirations

v2 - Payments & ratings flow completion - Org/Team spaces, multi-tenant - Route optimization and batched loads

---

14) Open Questions

- Payment model (per match, subscription, or free?)
- SMS provider (Twilio vs others) and cost caps
- Minimum viable trust surface (ID verify? company SSO?)

---

15) Appendix — MongoDB Index DDL (sketch)

```
// Users
users.createIndex({ email: 1 }, { unique: true });

// Carpool
carpoolrides.createIndex({ origin: '2dsphere' });
carpoolrides.createIndex({ destination: '2dsphere' });
carpoolrides.createIndex({ status: 1, date: 1 });

// Loads
loads.createIndex({ origin: '2dsphere' });
loads.createIndex({ destination: '2dsphere' });
loads.createIndex({ status: 1, 'deliveryWindow.start': 1 });

// TruckAvailability
truckavailability.createIndex({ route: '2dsphere' }); // as LineString
truckavailability.createIndex({ status: 1, availableFrom: 1, availableUntil:
1 });

// Matches
matches.createIndex({ type: 1, status: 1 });
matches.createIndex({ driverOrTruckId: 1 });

// Messages
messages.createIndex({ matchId: 1, timestamp: 1 });
```