iTarang Battery Monitoring Dashboard — Full Roadmap

Project: "Antigravity" — Samsara-style Battery Fleet Monitoring

---

1. Executive Summary

Build a real-time battery fleet monitoring dashboard within the existing iTarang CRM (Next.js 16 + React 19 + Supabase + Drizzle ORM) that provides Samsara-like visibility into your e-rickshaw battery fleet. The dashboard will track State of Charge (SOC), State of Health (SOH), voltage, current, charge cycles, temperature, GPS location, distance, and energy consumption — enabling proactive warranty management, dealer performance insights, and fleet health monitoring at scale.

Scale target: 22,000+ batteries × ~1 reading/minute = **~31.7 million data points/day**

---

2. Database Recommendation: TimescaleDB

Why TimescaleDB (not QuestDB, InfluxDB, or ClickHouse)

Criteria	TimescaleDB	QuestDB	InfluxDB 3.0	ClickHouse
PostgreSQL compatible	✅ Native extension	Partial (PGWire)	❌ New engine	❌ Own SQL dialect
Works with Drizzle ORM	✅ Same Postgres driver	❌ Cursor issues	❌ Different client	❌ Different client
Works with Supabase	✅ Extension available	❌ Separate DB	❌ Separate DB	❌ Separate DB
JOINs with CRM tables	✅ Same database	❌ Cross-DB needed	❌ Cross-DB needed	❌ Cross-DB needed
Continuous Aggregations	✅ Built-in	❌	❌	Manual MVs
Compression	✅ 90%+ for time-series	✅	✅	✅
Retention policies	✅ Built-in	✅	Limited (72h free)	Manual
Team learning curve	Minimal (it's Postgres)	Medium	High (new model)	High

The Killer Advantage

Your project already uses **Supabase (Postgres) + Drizzle ORM + `postgres` driver**. TimescaleDB is a Postgres

extension — meaning:

- **Same `DATABASE_URL`** — no new connection management
- **Same Drizzle schema definitions** — hypertables are just Postgres tables with automatic time partitioning
- **JOINs between CRM data and telemetry** — e.g., "show me SOH degradation for batteries sold by Dealer X"
- **No new infrastructure** — enable the extension on your existing Supabase instance (or a dedicated Supabase project for telemetry)

Setup Approach

Option A — Supabase Extension (Recommended for start)

```sql
CREATE EXTENSION IF NOT EXISTS timescaledb;
```

Enable TimescaleDB on your existing Supabase project, or create a dedicated Supabase project for telemetry to isolate high-write load from CRM queries.

Option B — Dedicated TimescaleDB Cloud (For 22K+ batteries) Use Timescale Cloud managed service alongside Supabase. Connect from Next.js API routes using a second Drizzle instance. This is better for production scale but adds operational complexity.

Recommendation: Start with Option A (same Supabase project, separate schema). Migrate to Option B when you hit performance limits (~5K-10K batteries).

---

3. Database Schema Design

3.1 Core Telemetry Tables (TimescaleDB Hypertables)

```sql
```

```sql
-- Enable extension
CREATE EXTENSION IF NOT EXISTS timescaledb;
CREATE EXTENSION IF NOT EXISTS postgis;  -- for GPS data

-- Schema separation
CREATE SCHEMA IF NOT EXISTS telemetry;

-- ============================================
-- BATTERY CAN DATA (Primary telemetry - ~1 reading/min)
-- Source: getcandata API
-- ============================================
CREATE TABLE telemetry.battery_readings (
    time            TIMESTAMPTZ    NOT NULL,
    device_id       VARCHAR(50)    NOT NULL,   -- e.g., "TK-51105-04HY"
    battery_id      VARCHAR(100),              -- FK to inventory.serial_number
    soc             REAL,                      -- State of Charge (%)
    soh             REAL,                      -- State of Health (%)
    voltage         REAL,                      -- Battery voltage (V)
    current         REAL,                      -- Current (A)
    charge_cycle    INTEGER,                   -- Charge cycle count
    temperature     REAL,                      -- Battery temp (°C)
    power_watts     REAL                       -- Computed: voltage × current
);

-- Convert to hypertable (auto-partitions by time)
SELECT create_hypertable('telemetry.battery_readings', 'time',
    chunk_time_interval => INTERVAL '1 day',
    if_not_exists => TRUE
);

-- Indexes for common query patterns
CREATE INDEX idx_battery_readings_device
    ON telemetry.battery_readings (device_id, time DESC);
CREATE INDEX idx_battery_readings_soc
    ON telemetry.battery_readings (device_id, soc, time DESC);

-- ============================================
-- GPS HISTORY (Location tracking)
-- Source: getgpshistory API
-- ============================================
CREATE TABLE telemetry.gps_readings (
    time            TIMESTAMPTZ    NOT NULL,
    device_id       VARCHAR(50)    NOT NULL,
    location        GEOGRAPHY(POINT, 4326),  -- PostGIS point
    latitude        REAL           NOT NULL,
    longitude       REAL           NOT NULL,
```

```
    altitude        REAL,
    speed           REAL,
    heading         REAL,
    device_battery  REAL,          -- IoT device battery voltage
    vehicle_battery REAL,          -- Vehicle battery voltage
    ignition_on     BOOLEAN,
    is_moving       BOOLEAN
);

SELECT create_hypertable('telemetry.gps_readings', 'time',
    chunk_time_interval => INTERVAL '1 day',
    if_not_exists => TRUE
);

CREATE INDEX idx_gps_device_time
    ON telemetry.gps_readings (device_id, time DESC);
CREATE INDEX idx_gps_location
    ON telemetry.gps_readings USING GIST (location);

-- ============================================
-- TRIP SUMMARIES (Daily/trip aggregates)
-- Source: getdistance API
-- ============================================
CREATE TABLE telemetry.trips (
    id              SERIAL PRIMARY KEY,
    device_id       VARCHAR(50)   NOT NULL,
    start_time      TIMESTAMPTZ   NOT NULL,
    end_time        TIMESTAMPTZ   NOT NULL,
    start_odometer  REAL,
    end_odometer    REAL,
    distance_km     REAL,
    start_location  GEOGRAPHY(POINT, 4326),
    end_location    GEOGRAPHY(POINT, 4326),
    last_ign_on     TIMESTAMPTZ,
    last_ign_off    TIMESTAMPTZ,
    created_at      TIMESTAMPTZ   DEFAULT NOW()
);

-- ============================================
-- ENERGY CONSUMPTION
-- Source: getfuelused API (repurposed for electric)
-- ============================================
CREATE TABLE telemetry.energy_consumption (
    id              SERIAL PRIMARY KEY,
    device_id       VARCHAR(50)   NOT NULL,
    start_time      TIMESTAMPTZ   NOT NULL,
    end_time        TIMESTAMPTZ   NOT NULL,
```

```
    energy_used_kwh REAL,          -- Converted from "fuel" litres
    start_soc       REAL,
    end_soc         REAL,
    last_ign_on     TIMESTAMPTZ,
    last_ign_off    TIMESTAMPTZ,
    charging_events JSONB DEFAULT '[]',
    created_at      TIMESTAMPTZ   DEFAULT NOW()
);
```

3.2 Continuous Aggregates (Pre-computed Rollups)

These run automatically and give you instant dashboard queries instead of scanning millions of rows.

sql

```sql
-- ============================================
-- HOURLY BATTERY SUMMARY
-- ============================================
CREATE MATERIALIZED VIEW telemetry.battery_hourly
WITH (timescaledb.continuous) AS
SELECT
    time_bucket('1 hour', time)    AS bucket,
    device_id,
    AVG(soc)                       AS avg_soc,
    MIN(soc)                       AS min_soc,
    MAX(soc)                       AS max_soc,
    AVG(voltage)                   AS avg_voltage,
    MIN(voltage)                   AS min_voltage,
    MAX(voltage)                   AS max_voltage,
    AVG(current)                   AS avg_current,
    AVG(temperature)               AS avg_temp,
    MAX(temperature)               AS max_temp,
    MAX(charge_cycle)              AS charge_cycle,
    AVG(soh)                       AS avg_soh,
    COUNT(*)                       AS reading_count
FROM telemetry.battery_readings
GROUP BY bucket, device_id
WITH NO DATA;

-- Auto-refresh every 30 minutes
SELECT add_continuous_aggregate_policy('telemetry.battery_hourly',
    start_offset    => INTERVAL '3 hours',
    end_offset      => INTERVAL '30 minutes',
    schedule_interval => INTERVAL '30 minutes'
);


-- ============================================
-- DAILY BATTERY SUMMARY
-- ============================================
CREATE MATERIALIZED VIEW telemetry.battery_daily
WITH (timescaledb.continuous) AS
SELECT
    time_bucket('1 day', time)     AS bucket,
    device_id,
    AVG(soc)                       AS avg_soc,
    MIN(soc)                       AS min_soc,
    MAX(soc)                       AS max_soc,
    AVG(voltage)                   AS avg_voltage,
    AVG(current)                   AS avg_current,
    AVG(temperature)               AS avg_temp,
    MAX(temperature)               AS max_temp,
```

```
    MAX(charge_cycle)           AS charge_cycle,
    MIN(soh)                    AS min_soh,
    COUNT(*)                    AS reading_count
FROM telemetry.battery_readings
GROUP BY bucket, device_id
WITH NO DATA;

SELECT add_continuous_aggregate_policy('telemetry.battery_daily',
    start_offset    => INTERVAL '3 days',
    end_offset      => INTERVAL '1 hour',
    schedule_interval => INTERVAL '1 hour'
);


-- ============================================
-- COMPRESSION POLICY (after 7 days, compress raw data)
-- ============================================
ALTER TABLE telemetry.battery_readings SET (
    timescaledb.compress,
    timescaledb.compress_segmentby = 'device_id',
    timescaledb.compress_orderby = 'time DESC'
);

SELECT add_compression_policy('telemetry.battery_readings', INTERVAL '7 days');

ALTER TABLE telemetry.gps_readings SET (
    timescaledb.compress,
    timescaledb.compress_segmentby = 'device_id',
    timescaledb.compress_orderby = 'time DESC'
);

SELECT add_compression_policy('telemetry.gps_readings', INTERVAL '7 days');


-- ============================================
-- RETENTION POLICY (keep raw data for 6 months, aggregates forever)
-- ============================================
SELECT add_retention_policy('telemetry.battery_readings', INTERVAL '6 months');
SELECT add_retention_policy('telemetry.gps_readings', INTERVAL '6 months');
```

3.3 CRM Bridge Tables (In your existing Supabase/Drizzle schema)

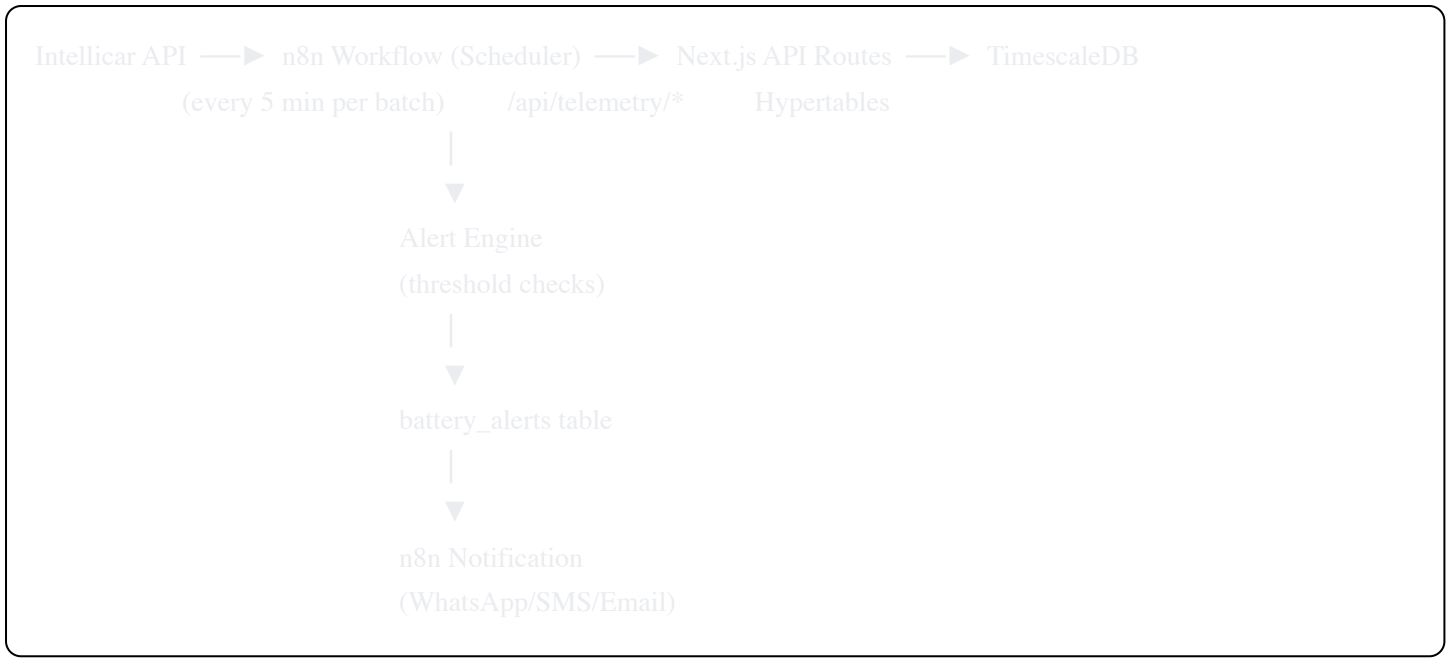
sql

```sql
-- Add to your existing schema.ts / migration
-- Links IoT devices to batteries in your inventory

CREATE TABLE device_battery_map (
    id              VARCHAR(255) PRIMARY KEY,
    device_id       VARCHAR(50)  NOT NULL UNIQUE,  -- Intellicar device ID
    battery_serial  VARCHAR(255) REFERENCES inventory(serial_number),
    vehicle_number  VARCHAR(50),                   -- e.g., "UP32 AT 1234"
    dealer_id       VARCHAR(255) REFERENCES accounts(id),
    lead_id         VARCHAR(255) REFERENCES leads(id),
    customer_name   TEXT,
    customer_phone  VARCHAR(20),
    activated_at    TIMESTAMPTZ DEFAULT NOW(),
    is_active       BOOLEAN DEFAULT TRUE,
    created_at      TIMESTAMPTZ DEFAULT NOW(),
    updated_at      TIMESTAMPTZ DEFAULT NOW()
);

CREATE INDEX idx_device_battery_dealer ON device_battery_map(dealer_id);
CREATE INDEX idx_device_battery_serial ON device_battery_map(battery_serial);

-- Alerts table for threshold breaches
CREATE TABLE battery_alerts (
    id              SERIAL PRIMARY KEY,
    device_id       VARCHAR(50)  NOT NULL,
    alert_type      VARCHAR(50)  NOT NULL,
    -- Types: low_soc, high_temp, soh_degradation, overcurrent,
    --        overvoltage, undervoltage, no_communication, deep_discharge
    severity        VARCHAR(20)  NOT NULL,  -- critical, warning, info
    message         TEXT         NOT NULL,
    reading_value   REAL,
    threshold_value REAL,
    acknowledged    BOOLEAN      DEFAULT FALSE,
    acknowledged_by UUID REFERENCES users(id),
    resolved_at     TIMESTAMPTZ,
    created_at      TIMESTAMPTZ  DEFAULT NOW()
);

CREATE INDEX idx_alerts_device ON battery_alerts(device_id, created_at DESC);
CREATE INDEX idx_alerts_unack ON battery_alerts(acknowledged, severity) WHERE NOT acknowledged;
```

---

4. Data Pipeline Architecture

4.1 Ingestion Flow



4.2 n8n Workflows to Build

Workflow	Trigger	Description
`fetch-can-data`	Cron (every 5 min)	Batch fetch CAN data for all active devices, insert into `battery_readings`
`fetch-gps-data`	Cron (every 5 min)	Batch fetch GPS history, insert into `gps_readings`
`fetch-daily-trips`	Cron (daily midnight)	Fetch distance/energy for all devices, insert into `trips` + `energy_consumption`
`battery-alert-check`	Cron (every 5 min)	Check latest readings against thresholds, create alerts
`alert-notify`	Webhook (on new alert)	Send WhatsApp/SMS notification via n8n integrations
`daily-fleet-report`	Cron (daily 8am)	Generate daily fleet health summary, email to stakeholders

4.3 API Routes to Build

```
src/app/api/telemetry/
├── ingest/
│   ├── can-data/route.ts       # POST - Bulk insert CAN readings
│   ├── gps-data/route.ts       # POST - Bulk insert GPS readings
│   ├── trips/route.ts          # POST - Insert trip summaries
│   └── energy/route.ts         # POST - Insert energy consumption
├── devices/
│   ├── route.ts                # GET - List all devices with latest status
│   ├── [deviceId]/
```

```
│   │   ├── route.ts            # GET - Device detail + latest readings
│   │   ├── readings/route.ts   # GET - Historical readings (with time range)
│   │   ├── gps/route.ts        # GET - GPS history
│   │   └── trips/route.ts      # GET - Trip history
├── fleet/
│   ├── overview/route.ts       # GET - Fleet-wide KPIs
│   ├── health/route.ts         # GET - SOH distribution, degradation
│   ├── alerts/route.ts         # GET/PATCH - Active alerts
│   └── map/route.ts            # GET - All device locations
├── analytics/
│   ├── soc-trends/route.ts     # GET - SOC patterns over time
│   ├── charge-cycles/route.ts  # GET - Charge cycle analysis
│   ├── dealer-comparison/route.ts # GET - Performance by dealer
│   └── warranty/route.ts       # GET - Warranty risk assessment
└── alerts/
    ├── route.ts                # GET - All alerts
    ├── [id]/route.ts           # PATCH - Acknowledge alert
    └── config/route.ts         # GET/PUT - Alert thresholds
```

---

5. Dashboard UI Design

5.1 Navigation Structure

Add to existing sidebar in `src/components/layout/sidebar.tsx`:

```
Battery Monitoring (new section)
├── Fleet Overview        — KPIs, map, alerts summary
├── Battery Health        — SOH tracking, degradation curves
├── Live Monitoring       — Real-time SOC/voltage/current
├── Trip Analytics        — Distance, energy efficiency
├── Alerts & Warranty     — Alert management, warranty flags
└── Device Management     — Device-battery mapping, activation
```

5.2 Page Breakdown

Page 1: Fleet Overview (`/battery-monitoring`)

Top Row — KPI Cards (4 cards)

Card	Value	Subtext
Total Active Batteries	1,847	+23 this week
Fleet Avg SOH	96.2%	▼ 0.3% from last month
Batteries Charging Now	312	16.9% of fleet
Active Alerts	7	3 critical, 4 warning

Middle Row — Two Panels

- **Left: Fleet Map** (full-width map showing all battery locations with color-coded pins)
  - 🟢 Green = SOC > 50%, healthy
  - 🟡 Yellow = SOC 20-50% or warning alert
  - 🔴 Red = SOC < 20% or critical alert
  - ⚫ Grey = Offline / no communication > 24h
- **Right: Alert Feed** (scrollable list of recent alerts with severity badges)

Bottom Row — Charts

- **SOC Distribution** (histogram: how many batteries at each SOC level right now)
- **Daily Distance Heatmap** (calendar view of fleet-wide daily km)
- **Charge Cycle Distribution** (bar chart of cycle counts across fleet)

Page 2: Battery Health (`/battery-monitoring/health`)

- **SOH Distribution** — Histogram showing fleet-wide SOH spread
- **SOH Degradation Curves** — Line chart: SOH vs charge cycles for each battery (overlay)
- **Warranty Risk Table** — Batteries with SOH < 80% flagged for warranty claim
- **Dealer-wise SOH Comparison** — Average SOH by dealer (identifies misuse patterns)
- **Temperature Impact Analysis** — SOH degradation correlated with operating temperature

Page 3: Live Monitoring (`/battery-monitoring/live/[deviceId]`)

Single battery deep-dive (like Samsara vehicle detail page)

- **Header**: Device ID, Battery Serial, Vehicle Number, Customer, Dealer
- **Real-time Gauges**: SOC (radial), Voltage, Current, Temperature
- **Time-series Charts** (selectable range: 1h, 6h, 24h, 7d, 30d):
  - SOC over time
  - Voltage over time

    - Current over time (shows charge/discharge patterns)
    - Temperature over time
- **Charge Session Timeline**: Visual blocks showing charge/discharge cycles
- **GPS Map**: Route trail for selected time period
- **Trip Table**: Recent trips with distance, energy used, efficiency

Page 4: Trip Analytics (`/battery-monitoring/trips`)

- **Fleet Distance Summary** — Total km today, this week, this month
- **Top 10 Highest Usage** — Batteries with most distance/energy consumed
- **Efficiency Ranking** — km per kWh by battery/dealer
- **Trip History Table** — Filterable by device, date range, dealer

Page 5: Alerts & Warranty (`/battery-monitoring/alerts`)

- **Alert Dashboard** — Active alerts grouped by severity
- **Alert Configuration** — Set thresholds per alert type
- **Warranty Tracker** — Batteries approaching warranty limits (SOH < 80%, high cycle count)
- **Alert History** — Full audit log of past alerts and resolutions

Page 6: Device Management (`/battery-monitoring/devices`)

- **Device-Battery Mapping** — Link Intellicar devices to battery serial numbers
- **Activation/Deactivation** — Toggle devices on/off
- **Bulk Import** — CSV upload for device mapping
- **Communication Status** — Last seen, data gap detection

---

6. Alert Engine Rules

Default Thresholds (configurable via UI)

Alert Type	Condition	Severity	Action
`low_soc`	SOC < 10%	Critical	Notify driver + dealer
`deep_discharge`	SOC = 0% for > 30 min	Critical	Warranty flag
`high_temp`	Temp > 55°C	Critical	Immediate notification
`soh_degradation`	SOH < 80%	Warning	Warranty review trigger

Alert Type	Condition	Severity	Action
`overcurrent`	Current > 100A	Warning	Log + notify if repeated
`overvoltage`	Voltage > 58.4V (for 48V system)	Warning	Check charger
`undervoltage`	Voltage < 42V	Critical	Deep discharge risk
`no_communication`	No data > 6 hours	Warning	Check device
`rapid_soh_drop`	SOH drops > 5% in 30 days	Critical	Warranty investigation
`excessive_cycles`	charge_cycle > 1500	Info	End-of-life planning

Data Quality Filters

Based on your sample data analysis, you have anomalous readings:

- Voltage spikes to 875,712V (clearly sensor error)
- Current spikes to 1,396,060A
- SOC readings > 100% (up to 123.71%)

Implement data quality checks in the ingestion pipeline:

```typescript
function isValidReading(reading: CANReading): boolean {
  return (
    reading.soc >= 0 && reading.soc <= 100 &&
    reading.battery_voltage >= 30 && reading.battery_voltage <= 70 && // 48V system range
    reading.current >= -50 && reading.current <= 200 &&              // reasonable current range
    (reading.battery_temp === null ||
     (reading.battery_temp >= -10 && reading.battery_temp <= 80))
  );
}
```

Log rejected readings to a `telemetry.rejected_readings` table for diagnostics.

---

7. Tech Stack Summary

Layer	Technology	Notes
Frontend	Next.js 16, React 19, Tailwind CSS	Existing stack
Charts	Recharts (existing) + add Tremor	Tremor for dashboard-grade cards/charts

Layer	Technology	Notes
Maps	Leaflet / react-leaflet OR Mapbox GL	Free tier works for fleet map
State Management	TanStack Query (existing)	Real-time polling every 30s for live page
API Layer	Next.js API Routes	Server-side, same project
ORM	Drizzle ORM (existing)	Works with TimescaleDB (it's Postgres)
Time-series DB	TimescaleDB (Postgres extension)	On Supabase or Timescale Cloud
CRM DB	Supabase Postgres (existing)	Same database, different schema
Data Pipeline	n8n (existing)	Scheduled workflows for Intellicar API
Notifications	n8n + WhatsApp/Twilio	Alert routing
Auth	Supabase Auth (existing)	Role-based access for battery views

New npm Packages to Add

```bash
npm install react-leaflet leaflet @tremor/react date-fns
npm install -D @types/leaflet
```

---

8. Implementation Checklist

Every task below is in build order — each depends on the ones above it. Work through top to bottom.

- [ ] 1. Enable TimescaleDB extension on Supabase (`CREATE EXTENSION IF NOT EXISTS timescaledb`)
- [ ] 2. Enable PostGIS extension on Supabase (`CREATE EXTENSION IF NOT EXISTS postgis`)
- [ ] 3. Create `telemetry` schema (`CREATE SCHEMA IF NOT EXISTS telemetry`)
- [ ] 4. Create `telemetry.battery_readings` hypertable (see Section 3.1)
- [ ] 5. Create `telemetry.gps_readings` hypertable
- [ ] 6. Create `telemetry.trips` table
- [ ] 7. Create `telemetry.energy_consumption` table
- [ ] 8. Create `telemetry.rejected_readings` table (for data quality rejects)
- [ ] 9. Create `telemetry.battery_hourly` continuous aggregate (see Section 3.2)
- [ ] 10. Create `telemetry.battery_daily` continuous aggregate
- [ ] 11. Set up compression policies (7-day threshold)
- [ ] 12. Set up retention policies (6 months raw, aggregates forever)
- [ ] 13. Create `device_battery_map` table in CRM schema (see Section 3.3)

- [ ] 14. Create `battery_alerts` table in CRM schema
- [ ] 15. Add Drizzle schema definitions for `device_battery_map` and `battery_alerts` in `schema.ts`
- [ ] 16. Create `src/lib/telemetry/types.ts` — TypeScript types for all telemetry data
- [ ] 17. Create `src/lib/telemetry/data-quality.ts` — `isValidReading()` filter + anomaly logging
- [ ] 18. Create `src/lib/telemetry/queries.ts` — reusable SQL query functions for hypertables
- [ ] 19. Create `src/lib/telemetry/db.ts` — telemetry DB connection (or reuse existing if same Supabase)
- [ ] 20. Create `src/lib/intellicar/types.ts` — API response types for all 4 Intellicar endpoints
- [ ] 21. Create `src/lib/intellicar/client.ts` — Intellicar API client (getcandata, getgpshistory, getdistance, getfuelused)
- [ ] 22. Build `POST /api/telemetry/ingest/can-data` — bulk insert CAN readings with data quality filter
- [ ] 23. Build `POST /api/telemetry/ingest/gps-data` — bulk insert GPS readings
- [ ] 24. Build `POST /api/telemetry/ingest/trips` — insert trip summaries
- [ ] 25. Build `POST /api/telemetry/ingest/energy` — insert energy consumption records
- [ ] 26. Install npm packages: `npm install react-leaflet leaflet @tremor/react date-fns` and `npm install -D @types/leaflet`
- [ ] 27. Build n8n workflow: `fetch-can-data` (cron every 5 min → Intellicar API → POST to ingest route)
- [ ] 28. Build n8n workflow: `fetch-gps-data` (cron every 5 min)
- [ ] 29. Build n8n workflow: `fetch-daily-trips` (cron daily midnight)
- [ ] 30. Seed initial `device_battery_map` entries (link existing Intellicar devices to battery serials)
- [ ] 31. Verify data is flowing: check `telemetry.battery_readings` count after 1 hour
- [ ] 32. Verify continuous aggregates are populating: check `telemetry.battery_hourly`
- [ ] 33. Add "Battery Monitoring" section to sidebar in `src/components/layout/sidebar.tsx`
- [ ] 34. Build `src/components/battery-monitoring/FleetKPICards.tsx` — total batteries, avg SOH, charging count, active alerts
- [ ] 35. Build `GET /api/telemetry/fleet/overview` — fleet-wide KPI aggregation query
- [ ] 36. Build `GET /api/telemetry/devices` — list all devices with latest reading (uses `DISTINCT ON`)
- [ ] 37. Build Fleet Overview page `src/app/(dashboard)/battery-monitoring/page.tsx` with KPI cards
- [ ] 38. Build `src/components/battery-monitoring/FleetMap.tsx` — Leaflet map with color-coded device pins
- [ ] 39. Build `GET /api/telemetry/fleet/map` — all device locations with latest SOC + alert status
- [ ] 40. Wire FleetMap into Fleet Overview page
- [ ] 41. Build `src/components/battery-monitoring/SOCDistribution.tsx` — histogram of current SOC levels
- [ ] 42. Build `src/components/battery-monitoring/AlertFeed.tsx` — scrollable recent alerts list
- [ ] 43. Build `GET /api/telemetry/alerts` — fetch alerts with filters (severity, acknowledged, device)
- [ ] 44. Wire SOC distribution + alert feed into Fleet Overview page
- [ ] 45. Build device list table on Fleet Overview with search, filter by dealer/status/SOC range
- [ ] 46. Build `GET /api/telemetry/devices/[deviceId]` — single device detail + latest readings
- [ ] 47. Build `GET /api/telemetry/devices/[deviceId]/readings` — historical readings with time range params
- [ ] 48. Build `src/components/battery-monitoring/BatteryGauge.tsx` — radial SOC gauge component
- [ ] 49. Build `src/components/battery-monitoring/SOCChart.tsx` — SOC time-series line chart (recharts)
- [ ] 50. Build `src/components/battery-monitoring/VoltageCurrentChart.tsx` — dual-axis voltage + current chart

- [ ] 51. Build Live Monitoring page `src/app/(dashboard)/battery-monitoring/live/[deviceId]/page.tsx`
- [ ] 52. Wire header (device ID, battery serial, customer, dealer) into live page
- [ ] 53. Wire real-time gauges (SOC, voltage, current, temp) into live page
- [ ] 54. Wire time-series charts with selectable range (1h, 6h, 24h, 7d, 30d) into live page
- [ ] 55. Set up TanStack Query polling (30s refetch) for live page data
- [ ] 56. Build `GET /api/telemetry/devices/[deviceId]/gps` — GPS history with time range
- [ ] 57. Build GPS route trail component on Leaflet map for single device
- [ ] 58. Build `src/components/battery-monitoring/ChargeTimeline.tsx` — visual charge/discharge session blocks
- [ ] 59. Build `GET /api/telemetry/devices/[deviceId]/trips` — trip history for device
- [ ] 60. Build `src/components/battery-monitoring/TripTable.tsx` — trip history data table
- [ ] 61. Wire GPS map + charge timeline + trip table into live page
- [ ] 62. Create `src/lib/telemetry/alert-engine.ts` — threshold checking logic for all alert types
- [ ] 63. Build `POST /api/telemetry/alerts` — create alert endpoint (called by alert engine)
- [ ] 64. Build `PATCH /api/telemetry/alerts/[id]` — acknowledge / resolve alert
- [ ] 65. Build `GET /api/telemetry/alerts/config` — fetch current alert thresholds
- [ ] 66. Build `PUT /api/telemetry/alerts/config` — update alert thresholds
- [ ] 67. Build n8n workflow: `battery-alert-check` (cron every 5 min → check latest readings → create alerts)
- [ ] 68. Build n8n workflow: `alert-notify` (webhook trigger → send WhatsApp/SMS via Twilio/n8n)
- [ ] 69. Build `src/components/battery-monitoring/AlertBadge.tsx` — severity indicator component
- [ ] 70. Build Alerts page `src/app/(dashboard)/battery-monitoring/alerts/page.tsx` — alert dashboard + config UI
- [ ] 71. Wire alert acknowledgment + resolution flow into alerts page
- [ ] 72. Build `GET /api/telemetry/fleet/health` — SOH distribution, degradation data
- [ ] 73. Build `src/components/battery-monitoring/SOHDegradationChart.tsx` — SOH vs charge cycles overlay
- [ ] 74. Build Battery Health page `src/app/(dashboard)/battery-monitoring/health/page.tsx`
- [ ] 75. Wire SOH distribution histogram into health page
- [ ] 76. Wire SOH degradation curves (SOH vs cycles per battery) into health page
- [ ] 77. Build `GET /api/telemetry/analytics/warranty` — batteries with SOH < 80% or high cycle count
- [ ] 78. Build warranty risk table on health page — flag batteries for warranty claims
- [ ] 79. Build `GET /api/telemetry/analytics/dealer-comparison` — avg SOH, alert frequency by dealer
- [ ] 80. Build `src/components/battery-monitoring/DealerComparisonChart.tsx` — bar chart by dealer
- [ ] 81. Wire dealer comparison into health page
- [ ] 82. Build Trip Analytics page `src/app/(dashboard)/battery-monitoring/trips/page.tsx`
- [ ] 83. Build fleet distance summary (today, this week, this month)
- [ ] 84. Build top 10 highest usage table (most distance/energy consumed)
- [ ] 85. Build efficiency ranking (km per kWh by battery/dealer)
- [ ] 86. Build `GET /api/telemetry/analytics/soc-trends` — SOC patterns over time
- [ ] 87. Wire SOC trend analysis into trip analytics or health page
- [ ] 88. Build Device Management page `src/app/(dashboard)/battery-monitoring/devices/page.tsx`
- [ ] 89. Build device-battery mapping UI (link Intellicar device to battery serial)

- [ ] 90. Build device activation/deactivation toggle
- [ ] 91. Build bulk CSV import for device-battery mapping
- [ ] 92. Build communication status view (last seen, data gap detection)
- [ ] 93. Add role-based access control — CEO sees full fleet, dealer sees only their batteries
- [ ] 94. Add role-based filtering to all telemetry API routes (check user.dealer_id)
- [ ] 95. Build n8n workflow: `daily-fleet-report` (cron 8am → generate summary → email stakeholders)
- [ ] 96. Add export functionality — PDF/XLSX download for fleet health report
- [ ] 97. Tune compression policies based on actual data volume
- [ ] 98. Add API response caching (Next.js `revalidate` or Redis) for fleet overview
- [ ] 99. Test mobile responsiveness on all dashboard pages
- [ ] 100. Load test with simulated 22K device data — verify query performance on continuous aggregates

---

9. Key Architectural Decisions

9.1 Polling vs WebSocket for Real-time

Recommendation: Start with polling (TanStack Query + 30s refetch), add WebSocket later.

At 22K batteries, real-time WebSocket for every reading creates significant server load. Polling the continuous aggregate (hourly/daily) is much cheaper. Only the single-battery live page benefits from fast polling (5-10s).

9.2 Separate Supabase Project for Telemetry?

Recommendation: Same project initially, separate when needed.

Advantages of same project: JOINs between CRM and telemetry, simpler deployment, single auth. Split when write volume impacts CRM read performance.

9.3 Intellicar API Rate Limits

You'll need to batch API calls. With 22K devices at 5-min intervals, that's ~4,400 calls/minute. Work with Intellicar on batch endpoints or implement pagination with n8n's loop nodes.

9.4 Data Anomaly Handling

Your sample data shows sensor errors (voltage > 800K, SOC > 100%). These MUST be filtered before storage to prevent chart scaling issues and false alerts. Log rejected readings separately for device diagnostics.

---

10. Sample Drizzle Schema Addition

Add this to `src/lib/db/schema.ts`:

```typescript
```

```
import { pgTable, varchar, real, integer, boolean,
         timestamp, serial, jsonb, uuid, index } from 'drizzle-orm/pg-core';

// ====== DEVICE-BATTERY MAPPING ======
export const deviceBatteryMap = pgTable('device_battery_map', {
  id: varchar('id', { length: 255 }).primaryKey(),
  device_id: varchar('device_id', { length: 50 }).notNull().unique(),
  battery_serial: varchar('battery_serial', { length: 255 }),
  vehicle_number: varchar('vehicle_number', { length: 50 }),
  dealer_id: varchar('dealer_id', { length: 255 }),
  lead_id: varchar('lead_id', { length: 255 }),
  customer_name: varchar('customer_name', { length: 255 }),
  customer_phone: varchar('customer_phone', { length: 20 }),
  activated_at: timestamp('activated_at', { withTimezone: true }).defaultNow(),
  is_active: boolean('is_active').default(true),
  created_at: timestamp('created_at', { withTimezone: true }).defaultNow(),
  updated_at: timestamp('updated_at', { withTimezone: true }).defaultNow(),
}, (table) => [
  index('idx_device_battery_dealer').on(table.dealer_id),
  index('idx_device_battery_serial').on(table.battery_serial),
]);

// ====== BATTERY ALERTS ======
export const batteryAlerts = pgTable('battery_alerts', {
  id: serial('id').primaryKey(),
  device_id: varchar('device_id', { length: 50 }).notNull(),
  alert_type: varchar('alert_type', { length: 50 }).notNull(),
  severity: varchar('severity', { length: 20 }).notNull(),
  message: varchar('message', { length: 500 }).notNull(),
  reading_value: real('reading_value'),
  threshold_value: real('threshold_value'),
  acknowledged: boolean('acknowledged').default(false),
  acknowledged_by: uuid('acknowledged_by'),
  resolved_at: timestamp('resolved_at', { withTimezone: true }),
  created_at: timestamp('created_at', { withTimezone: true }).defaultNow(),
}, (table) => [
  index('idx_alerts_device').on(table.device_id, table.created_at),
]);

// NOTE: The telemetry hypertables (battery_readings, gps_readings, etc.)
// are created via raw SQL migration since Drizzle doesn't support
// TimescaleDB hypertable creation. Query them using Drizzle's sql`` tag
// or raw queries through the postgres driver.
```

11. File Structure for New Dashboard Module

```
src/
├── app/(dashboard)/battery-monitoring/
│   ├── page.tsx                    # Fleet Overview
│   ├── health/page.tsx             # Battery Health
│   ├── live/[deviceId]/page.tsx    # Single Battery Live
│   ├── trips/page.tsx              # Trip Analytics
│   ├── alerts/page.tsx             # Alerts & Warranty
│   └── devices/page.tsx            # Device Management
├── app/api/telemetry/
│   ├── ingest/can-data/route.ts
│   ├── ingest/gps-data/route.ts
│   ├── ingest/trips/route.ts
│   ├── ingest/energy/route.ts
│   ├── devices/route.ts
│   ├── devices/[deviceId]/route.ts
│   ├── devices/[deviceId]/readings/route.ts
│   ├── devices/[deviceId]/gps/route.ts
│   ├── fleet/overview/route.ts
│   ├── fleet/health/route.ts
│   ├── fleet/map/route.ts
│   ├── analytics/soc-trends/route.ts
│   ├── analytics/dealer-comparison/route.ts
│   ├── analytics/warranty/route.ts
│   └── alerts/route.ts
├── components/battery-monitoring/
│   ├── FleetMap.tsx                # Leaflet map with device pins
│   ├── BatteryGauge.tsx            # Radial SOC gauge
│   ├── SOCChart.tsx                # SOC time-series chart
│   ├── VoltageCurrentChart.tsx     # Dual-axis V/A chart
│   ├── SOHDegradationChart.tsx     # SOH vs cycles
│   ├── ChargeTimeline.tsx          # Charge session blocks
│   ├── AlertFeed.tsx               # Scrollable alert list
│   ├── AlertBadge.tsx              # Severity indicator
│   ├── DeviceStatusCard.tsx        # Single device summary card
│   ├── FleetKPICards.tsx           # Top-level KPI row
│   ├── SOCDistribution.tsx         # Histogram
│   ├── DealerComparisonChart.tsx   # Bar chart by dealer
│   └── TripTable.tsx               # Trip history data table
└── lib/
    ├── telemetry/
    │   ├── db.ts                   # Telemetry DB connection (if separate)
    │   ├── queries.ts              # Common telemetry SQL queries
    │   ├── types.ts                # TypeScript types for telemetry data
    │   ├── data-quality.ts         # Validation/filtering functions
```

```
│   └── alert-engine.ts          # Alert threshold checking logic
└── intellicar/
    ├── client.ts                # Intellicar API client
    └── types.ts                 # API response types
```

---

12. Improvement Suggestions Over Current Setup

1. **Data Quality Layer** — Your sample data has sensor errors (voltage 800K+, SOC 123%). Build a robust validation pipeline BEFORE storage. This is critical for reliable dashboards.
2. **Computed Metrics** — Store `power_watts` (V × A) at ingestion time. Pre-compute daily energy consumption (kWh) instead of calculating at query time.
3. **Device Health Monitoring** — Your `battery_temp` is null in 99.99% of readings. Track which sensors are failing and alert on data quality, not just battery health.
4. **Dealer Performance Scoring** — Use battery health data to create a dealer quality score (avg SOH, alert frequency, misuse patterns). This is a unique value prop for your CRM.
5. **Predictive Analytics (Phase 2)** — Once you have 6+ months of data, train a simple regression model to predict battery EOL date based on SOH degradation rate, cycle count, and temperature history.
6. **Mobile App Push** — For drivers/dealers, push notifications when SOC is critically low or when anomalies are detected. This prevents "bricked" e-rickshaws.
7. **Warranty Automation** — Auto-generate warranty claim documents when SOH < 80% before expected lifecycle. This ties directly into your existing CRM workflow.

---

13. Estimated Effort

Milestone	Tasks	Duration	Developer(s)
DB + Ingestion Pipeline	#1 – #32	2 weeks	1 Full-stack
Fleet Overview Dashboard	#33 – #45	2 weeks	1 Full-stack + 1 Frontend
Single Battery Live View	#46 – #61	2 weeks	1 Full-stack + 1 Frontend
Alert Engine	#62 – #71	1.5 weeks	1 Full-stack
Health & Analytics	#72 – #87	2 weeks	1 Full-stack + 1 Frontend
Device Mgmt + RBAC + Polish	#88 – #100	2.5 weeks	1 Full-stack
Total	**100 tasks**	**~12 weeks**	**2 developers**

Can be compressed to 8 weeks with 3 developers working in parallel (backend/pipeline, frontend/UI, n8n/integrations).