iTarang Battery Dashboard

The Complete Beginner's Guide

Every concept in YOUR codebase, explained like you're in high school

GitHub	github.com/apoorvgupta0709/intellicardashboard
Stack	Next.js 16 + React 18 + TypeScript + Tailwind CSS
Database	PostgreSQL + Drizzle ORM + Supabase
Charts	Recharts + Leaflet Maps
Scale	22,000 batteries x 31.7M data points/day

This tutorial includes **6 interactive Excalidraw diagrams** (available in the companion chat) and **18 chapters** covering every technology used in the project.

How to use: Read top to bottom. Every code example is from YOUR actual codebase. The diagram panels reference the interactive Excalidraw visuals from the chat — keep them open side by side.

Table of Contents

1 What Is This Project?

1 DIAGRAM: What is the Intellicar Dashboard?

See Excalidraw Diagram 1 — Shows the full journey from e-rickshaw to dashboard screen

You sell lithium-ion batteries for e-rickshaws. You have 22,000+ of them driving around India. Each battery has a tiny computer (IoT device by Intellicar) glued to it that reports data **every 60 seconds**.

Your dashboard is like a **principal's office** that shows what every student (battery) is doing:

What You Want to Know	Battery Equivalent
Is the student in class?	Is the battery charged? (SOC)
Is the student healthy?	Is the battery degrading? (SOH)
Where is the student?	GPS location on map
Is something wrong?	Alerts (overheating, dead battery)
Semester performance?	Charge cycles, trip history

Scale problem: 22,000 batteries x 1 reading/minute = **31.7 million data points per day.** That's like getting 31 million WhatsApp messages daily and needing to organize them all.

What data comes from each battery

Data Point	What It Means	Normal Range
SOC	State of Charge — how FULL	0-100%
SOH	State of Health — how HEALTHY vs new	100% (new) to 80% (replace)
Voltage	Electrical pressure (48V system)	42V (empty) to 58V (full)
Current	Electricity flowing	Negative=discharging, Positive=charging
Temperature	How hot the battery is	25-45C normal, >55C = DANGER
Charge Cycle	How many full charges	0 (new) to 1500+ (end of life)
GPS	Location coordinates	Latitude + Longitude
Speed / Trip	Movement and distance	km/h, total km traveled

2 The Tech Stack

2 DIAGRAM: Your Tech Stack — The Team

See Excalidraw Diagram 2 — Shows frontend vs backend tools with analogies

Think of building an app like making a Bollywood movie. Each tool has ONE job:

Tool	Movie Analogy	What It Does
Next.js 16	Director	Controls everything — pages, URLs, routing
React 18	Actors	Each button, card, chart is a reusable piece
TypeScript	Script	Tells actors exactly what to say (catches typos)
Tailwind CSS	Costume Dept	Makes everything look good with class names
PostgreSQL	Film Archive	Stores all 31.7M daily data points
Drizzle ORM	Librarian	Fetches data without writing raw SQL
Supabase	The Studio	Hosts your database in the cloud
Recharts	Graphics Dept	Draws bar charts, line charts
Leaflet	Map Dept	Shows battery locations on India map
Heroicons	Props Dept	Pretty icons for buttons and menus

3 File Structure

3 **DIAGRAM: Project Folder Structure — The Map**

See Excalidraw Diagram 3 — Shows where every file lives and what URL it creates

Your project has **3 main folders** inside `src/`:

src/app/ — Pages and API Routes

The folder names literally become the URLs:

File Path	URL Created	What It Shows
app/page.tsx	/	Redirects to /battery-monitoring
(dashboard)/battery-monitoring/page.tsx	/battery-monitoring	Fleet Overview
battery-monitoring/health/page.tsx	/battery-monitoring/health	Battery Health
battery-monitoring/alerts/page.tsx	/battery-monitoring/alerts	Alerts & Rules
battery-monitoring/trips/page.tsx	/battery-monitoring/trips	Trip Analytics
battery-monitoring/devices/page.tsx	/battery-monitoring/devices	Device Management
battery-monitoring/live/[deviceId]/page.tsx	/battery-monitoring/live/TK-51105	Single battery view
api/telemetry/fleet/overview/route.ts	GET /api/.../overview	Returns KPI JSON
api/telemetry/ingest/can-data/route.ts	POST /api/.../can-data	Accepts new readings

Two naming tricks: *(dashboard) — parentheses mean 'group pages but don't add to URL'. [deviceId] — brackets mean 'this part of the URL is a variable'.*

src/components/ — Reusable LEGO Blocks

Component	What It Draws
FleetKPICards.tsx	5 stat cards (Active Batteries, Avg SOH, etc.)
BatteryGauge.tsx	Circular SOC meter (like a speedometer)
FleetMap.tsx	Leaflet map with colored pins
SOCDistribution.tsx	Bar chart of SOC levels
AlertFeed.tsx	Scrolling list of recent alerts
SOHDegradationChart.tsx	Health decline over time
TripHistoryTable.tsx	Past trips with distance/energy

src/lib/ — The Brain (no UI)

File	What It Does
db/schema.ts	Defines database table structure using Drizzle ORM
telemetry/db.ts	Creates the database connection
telemetry/types.ts	Defines the "shape" of data (TypeScript interfaces)
telemetry/queries.ts	SQL queries to insert/read telemetry data
telemetry/data-quality.ts	Filters out garbage sensor readings (875,712V!)
telemetry/alert-engine.ts	Checks readings against thresholds, creates alerts
auth/AuthContext.tsx	Client-side role switching (CEO vs Dealer)
intellicar/client.ts	HTTP client for Intellicar hardware API

4 TypeScript — Labels on Everything

The Problem TypeScript Solves

Regular JavaScript lets you put anything anywhere. TypeScript adds labels — and catches mistakes **instantly** instead of 3 weeks later when a customer complains.

```
// JavaScript — no labels, chaos
let battery = { charge: 75 };
console.log(battery.chrage);  // Typo! But JS says nothing. Shows "undefined".

// TypeScript — everything labeled
interface Battery { charge: number; }
let battery: Battery = { charge: 75 };
console.log(battery.chrage);  // INSTANT ERROR: "Did you mean 'charge'?"
```

YOUR CODE — src/lib/telemetry/types.ts

```
export interface CANReading {
    time: Date;                // MUST exist, MUST be a Date
    device_id: string;         // MUST exist, MUST be text
    soc: number | null;        // Either a number OR nothing
    soh: number | null;        // (sensor might fail)
    voltage: number | null;
    current: number | null;
    temperature: number | null;
    power_watts: number | null;
}
```

Reading the code like English:

- `number | null` = 'either a number OR nothing' (the | means 'or')
- `string` = text (like 'TK-51105')
- `export` = 'other files can import this'
- ? after a name = 'this field is optional'

Generics — Fill in the Blank

```
// In intellicar/client.ts:
async function postToIntellicar<T>(endpoint: string, payload: any): Promise<T>

// When you call it, T gets replaced:
const vehicles = await postToIntellicar<any[]>('listvehicledevicemapping', {});
//                                      ^^^^^ T = any[] (an array)
```

Think of <T> like a Mad Libs blank: 'I want to fetch _____ from the API.' You fill it in when you call the function.

5 React — Building the UI

4 **DIAGRAM: React — How Your UI Actually Works**

See Excalidraw Diagram 4 — Components, State, useEffect, Server vs Client, Context

Concept 1: Components = LEGO Blocks

Every piece of UI is a **function that returns HTML-like code (JSX)**. You can reuse it anywhere:

```
// BatteryGauge — a reusable circular meter
export default function BatteryGauge({ soc, size = 160, label = "SOC" }) {
    return (
        <div>
            <svg> {/* draws the circle */} </svg>
            <span>{Math.round(soc)}%</span>
        </div>
    );
}
// Use it: <BatteryGauge soc={75} />  or  <BatteryGauge soc={20} size={100} />
```

Concept 2: State = Memory That Triggers Re-draw

```
const [data, setData] = useState({
    activeBatteries: 0,     // Initial value
    avgSoh: "0.0",
});

// Later, when API responds:
setData({ activeBatteries: 147, avgSoh: "96.2" });
// React AUTOMATICALLY redraws the cards with new numbers!
```

Analogy: A regular variable is writing on a whiteboard — erasing doesn't notify anyone. useState is a projector connected to a computer — change the data, everyone sees it.

Concept 3: useEffect = 'When page loads, do this'

```
useEffect(() => {
    fetch('/api/telemetry/fleet/overview')
        .then(res => res.json())
        .then(json => setData(json));  // Update state with API data
}, []);   // Empty [] = run ONCE on load
```

What's in the []	When It Runs
[] (empty)	Once, when component first loads

What's in the []	When It Runs
[count]	Every time count changes
Nothing at all	Every render (usually a mistake!)

Concept 4: Server vs Client Components

Feature	Server Component	Client Component
Keyword	(default, no keyword)	"use client" at line 1
Runs on	Server only	Server first, then browser
Can use	Database, env secrets	useState, useEffect, onClick
Speed	Super fast (no JS sent)	Slower (JS must download)
Your files	page.tsx files	Most files in components/

Rule: *If your component needs useState, useEffect, or onClick — it MUST have 'use client'. Otherwise, leave it as server component (faster).*

Concept 5: Context = Shared Bulletin Board

Without Context, passing data from grandparent to grandchild is like telephone. **Context is a bulletin board** — anyone can read it. Your app uses AuthContext so the Sidebar, KPI Cards, and API routes all know 'am I CEO or Dealer?' without passing props through 10 layers.

6 Next.js — The Framework

URL Routing = Folder Names

Next.js uses your **file system** as the router. Create a folder = create a URL:

```
YOUR FOLDER:  src/app/(dashboard)/battery-monitoring/health/page.tsx
BECOMES URL:  /battery-monitoring/health
```

Layouts — Shared Wrappers

The dashboard layout (sidebar + header) wraps ALL pages under /battery-monitoring/*. When you click 'Health' in the sidebar, **only the right panel changes**. The sidebar stays put — no flickering!

API Routes — Your Backend

```
// api/telemetry/fleet/overview/route.ts
// Creates: GET /api/telemetry/fleet/overview

export async function GET(req: Request) {
    const auth = await getServerSession(req);     // 1. WHO is asking?
    const result = await telemetryDb.execute(sql`  // 2. QUERY database
        SELECT COUNT(*) as active_count
        FROM device_battery_map WHERE is_active = TRUE
    `);
    return NextResponse.json({                     // 3. RETURN answer
        activeBatteries: Number(result[0]?.active_count || 0),
    });
}
```

Dynamic Import — Loading Heavy Stuff Lazily

```
// FleetMapDynamic.tsx
const FleetMapContext = dynamic(
    () => import('./FleetMap'),  // Load this file...
    { ssr: false }               // ...ONLY in the browser (not server!)
);
```

Why? Leaflet needs the browser's 'window' object. Servers don't have 'window'. Without ssr: false, the server crashes.

7 Tailwind CSS — Styling Without CSS Files

Instead of writing separate .css files, Tailwind uses class names directly in your JSX:

```
// Traditional: Create styles.css, define .card, import it
// Tailwind:
<div className="bg-white rounded-xl border border-slate-200 p-5">
```

Reading Tailwind Like English

Tailwind Class	What It Does	CSS Equivalent
bg-white	White background	background-color: white
rounded-xl	Rounded corners (large)	border-radius: 12px
p-5	Padding all sides	padding: 20px (5 x 4px)
text-sm	Small text	font-size: 14px
font-bold	Bold text	font-weight: 700
flex	Flexbox container	display: flex
grid grid-cols-3	3-column grid	grid-template-columns: repeat(3, 1fr)
lg:col-span-2	On large screens, span 2 cols	@media (min-width: 1024px) ...
hover:bg-blue-100	Change bg on hover	:hover { ... }
animate-pulse	Pulsing animation	Loading skeleton effect

The **lg:** prefix means 'only apply on large screens.' That's responsive design in one word! On mobile everything stacks (1 col), on desktop it splits (3 cols).

8 The Database

5 **DIAGRAM: Your Database — What Tables Look Like**

See Excalidraw Diagram 5 — Actual table structures with sample rows, showing how device_id connects eve

Table 1: device_battery_map — Who's Who

Links an IoT device to a physical battery, customer, and dealer:

device_id	battery_serial	vehicle_number	customer_name	dealer_id	is_active
TK-51105	BAT-001234	UP32-AB-1234	Ram Kumar	DLR-007	true
TK-51106	BAT-005678	UP32-CD-5678	Shyam Singh	DLR-012	true
TK-51107	BAT-009012	UP32-EF-9012	Mohan Lal	DLR-007	false

Table 2: telemetry.battery_readings — Sensor Data

Every 60 seconds, each battery sends a reading. **22,000 new rows every minute!**

time	device_id	soc	soh	voltage	current	temp	cycles
2025-03-01 10:00	TK-51105	75.0	96.2	51.2	-12.5	32	245
2025-03-01 10:01	TK-51105	74.8	96.2	51.1	-13.1	33	245
2025-03-01 10:00	TK-51106	42.0	89.5	48.7	+5.2	28	512

Table 3: battery_alerts — Problems Detected

device_id	alert_type	severity	message	acknowledged
TK-51105	High Temp	critical	Critical temp: 58C	false
TK-51106	Low SOC	warning	Battery low: 12%	false
TK-51107	Low SOC	critical	Critically low: 3%	true

How Tables Connect

The `device_id` column appears in ALL three tables. It's the **thread** that ties everything together — like a student ID that connects across all school systems.

Drizzle ORM — Your Code Uses BOTH Approaches

- **Drizzle ORM** for CRM tables (device_battery_map, battery_alerts) — type-safe
- **Raw SQL** for telemetry tables (battery_readings) — because Drizzle doesn't support TimescaleDB features

SECURITY: *Always use Drizzle's sql`` tag for raw queries. Never concatenate strings into SQL — that's how hackers steal data (SQL injection)!*

9 The Data Pipeline

6 **DIAGRAM: The Data Pipeline — Battery to Screen**

See Excalidraw Diagram 4 — 10-step journey including quality filter and alert engine

Step 1: Intellicar API Client (Token Caching)

Instead of getting a new API token for every call (slow!), we cache it and reuse for 55 minutes:

```
let cachedToken: string | null = null;
let tokenExpiresAt: number = 0;

export async function getIntellicarToken(): Promise<string> {
    if (cachedToken && Date.now() < tokenExpiresAt) {
        return cachedToken;  // FREE! No API call needed.
    }
    // Otherwise, get a NEW token...
    cachedToken = data.data.token;
    tokenExpiresAt = Date.now() + 3600 * 1000 - 300000; // 55 min
}
```

Step 2: Data Quality Filter

Your real sensor data has glitches. This filter catches them:

Bad Reading	Why It's Bad	Filter Action
SOC = 123.71%	Battery can't be >100%	REJECTED
Voltage = 875,712V	48V system can't produce 875KV	REJECTED
Current = 1,396,060A	More than a lightning bolt!	REJECTED
Temperature = null	Sensor broken (99.99% are null)	Allowed (null = unknown)

Step 3: Alert Engine

Alert Type	Condition	Severity
High Temperature	temp >= 55C	CRITICAL
High Temperature	temp >= 45C	Warning
Low SOC	SOC <= 5%	CRITICAL
Low SOC	SOC <= 15%	Warning
Low Voltage	voltage < 42V	CRITICAL

Deduplication: If battery TK-51105 is at 60C and we check every 5 min, we don't want 288 alerts/day. The engine checks if an unacknowledged alert already exists before creating a new one.

Step 4: Ingestion API — The Entire Pipeline in One Function

```
export async function POST(req: Request) {
    const payloads = await req.json();                    // 1. Read data
    const { valid, rejected } = filterCANBatch(payloads);  // 2. Filter
    if (valid.length > 0) {
        await insertCANReadings(valid as CANReading[]);    // 3. Store
        await processAlerts(valid as CANReading[]);        // 4. Alert
    }
    return NextResponse.json({                              // 5. Respond
        inserted: valid.length, rejected: rejected.length,
    });
}
```

10 SQL Patterns in Your Project

Pattern 1: DISTINCT ON — Latest reading per device

```
SELECT DISTINCT ON (device_id)
    device_id, time, soc, soh, voltage
FROM telemetry.battery_readings
ORDER BY device_id, time DESC
```

For each unique device_id, keeps **only the most recent row**. This is how Fleet Overview gets 'current SOC' without scanning all history.

Pattern 2: WITH (CTE) — Multi-step queries

```
WITH LatestReadings AS (
    SELECT DISTINCT ON (device_id) soh, device_id
    FROM telemetry.battery_readings
    ORDER BY device_id, time DESC
)
SELECT AVG(soh) as avg_soh FROM LatestReadings
WHERE soh IS NOT NULL
```

CTE = Common Table Expression. Like solving math in steps: first find each battery's latest SOH, then average them.

Pattern 3: Conditional WHERE — Role-based filtering

```
${auth.role === 'dealer'
    ? sql`AND dealer_id = ${auth.dealer_id}`  // Dealer: only THEIR batteries
    : sql``}                                   // CEO: sees EVERYTHING
```

If you're a dealer, the query adds a filter. If you're CEO, it adds nothing.

11 Recharts — Drawing Charts

```
<ResponsiveContainer width="100%" height="100%">
    <BarChart data={data}>
        <XAxis dataKey="range" />     {/* X-axis: "0-20%", "21-40%" */}
        <YAxis />                      {/* Y-axis: auto numbers */}
        <Tooltip />                    {/* Info on hover */}
        <Bar dataKey="count">          {/* Bar height = count value */}
            {data.map((_, i) => <Cell key={i} fill={COLORS[i]} />)}
        </Bar>
    </BarChart>
</ResponsiveContainer>
```

Give Recharts an array of objects. Tell it which field is X (`dataKey='range'`) and which is Y (`dataKey='count'`). `ResponsiveContainer` makes the chart resize with the window.

12 Leaflet — Interactive Maps

Leaflet is split into two files because of the SSR problem. `FleetMapDynamic.tsx` loads `FleetMap.tsx` only in the browser using `dynamic({ ssr: false })`.

```
<MapContainer center={[26.8, 80.9]} zoom={6}>
    <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
    {devices.map(device => (
        <Marker position={[device.lat, device.lng]}>
            <Popup>{device.name} -- SOC: {device.soc}%</Popup>
        </Marker>
    ))}
</MapContainer>
```

13 Authentication — Who Sees What

Role	What They See	Real-World Analogy
CEO	ALL batteries, ALL dealers	Principal sees all students
Dealer	Only THEIR batteries	Teacher sees only their class

Client side: `AuthContext.tsx` stores role in localStorage. Server side: every API route calls `getServerSession(req)` and adds WHERE clauses based on role.

14 Environment Variables — Secrets

Prefix	Who Can See	Example
No prefix	Server only (safe!)	DATABASE_URL, INTELLICAR_PASSWORD
NEXT_PUBLIC_	EVERYONE (browsers too!)	NEXT_PUBLIC_MAP_URL

CRITICAL: *Never put NEXT_PUBLIC_ on database URLs or API keys. Anything with that prefix gets embedded in JavaScript that browsers download — hackers can see it!*

15 The Full Request Lifecycle

7 **DIAGRAM: What Happens When You Open the Dashboard?**

See Excalidraw Diagram 6 — 10-step sequence from URL to rendered dashboard

When you type `/battery-monitoring` in your browser:

Step	What	Where	Time
1	You type the URL	Browser	0ms
2	Next.js finds page.tsx	Server	~5ms
3	Server renders HTML skeleton	Server	~50ms
4	Browser shows skeleton	Browser	~200ms
5	FleetKPICards mounts, useEffect fires	Browser	~210ms
6	fetch(/api/telemetry/fleet/overview)	Network	~215ms
7	API route runs 4 SQL queries	Server+DB	~300ms
8	JSON response returns	Network	~400ms
9	setData() triggers React re-render	Browser	~410ms
10	User sees the full dashboard!	Screen	~500ms

Key insight: The HTML skeleton appears in ~200ms (instant!). Data fills in ~300ms later. Users feel it's instant because the layout shows immediately.

16 Patterns Cheat Sheet

Pattern	Where Used	What It Does
useState	FleetKPICards, AlertFeed	Memory that triggers re-draw
useEffect + fetch	FleetKPICards, AlertFeed	Load data on page load
"use client"	Most components	Needs browser features
dynamic({ ssr: false })	FleetMapDynamic	Only load in browser
(parentheses) folder	(dashboard)	Group without changing URL
[brackets] folder	[deviceId]	Variable URL segment
sql`` tag	All API routes	Safe SQL (prevents injection)
DISTINCT ON	Fleet overview	Latest reading per device
WITH ... AS	SOH averaging	Multi-step SQL
Conditional sql	Role-filtered queries	Different data per role
Token caching	intellicar/client.ts	Reuse API tokens
Data quality filter	data-quality.ts	Reject bad readings
Alert deduplication	alert-engine.ts	No duplicate alerts

17 Glossary

Term	Meaning	Simple Analogy
SOC	State of Charge (0-100%)	Phone battery %
SOH	State of Health (100%=new)	How worn out the battery is
CAN	Controller Area Network	Language batteries speak
API	Application Programming Interface	A waiter taking orders
JSON	JavaScript Object Notation	Data format: { "name": "Ram" }
ORM	Object-Relational Mapping	Translator: code to SQL
SSR	Server-Side Rendering	Drawing page on server first
Hydration	Making server HTML interactive	Waking up static HTML
CTE	Common Table Expression	Named temp result in SQL
RBAC	Role-Based Access Control	Different people see different data
IoT	Internet of Things	Physical devices on internet
Epoch	Unix timestamp	Seconds since Jan 1, 1970

18 Your 5 Practice Challenges

Now that you understand every piece, try these exercises. Each one uses concepts from this tutorial:

Challenge 1: Add 'Average Voltage' as a 6th KPI Card

Difficulty: Easy

Modify the fleet/overview API route to add a voltage query. Add a new entry to kpiConfig in FleetKPICards.tsx.

Uses: *Chapters 5 (useState), 6 (API routes), 8 (SQL)*

Challenge 2: Make SOCDistribution fetch REAL data

Difficulty: Medium

Replace mockData in SOCDistribution.tsx. Create a new API route at api/telemetry/analytics/soc-distribution/route.ts that groups batteries by SOC range.

Uses: *Chapters 5 (useEffect), 6 (API routes), 10 (SQL GROUP BY)*

Challenge 3: Add search to the Devices page

Difficulty: Medium

Add a text input that filters the device list by vehicle number or customer name. Use URL search params and the use-debounce package.

Uses: *Chapters 5 (useState), 6 (routing), 7 (Tailwind)*

Challenge 4: Add 'Rapid SOH Drop' alert type

Difficulty: Hard

In alert-engine.ts, add logic: if a battery's SOH drops >5% in 30 days, create a critical alert. Query the 30-day-old reading and compare.

Uses: *Chapters 9 (alert engine), 10 (SQL with date math)*

Challenge 5: Add CSV export to Trips page

Difficulty: Hard

Add a 'Download CSV' button that fetches trip data and triggers a browser download as a .csv file.

Uses: *Chapters 5 (onClick), 6 (API route), browser Blob API*

Each challenge uses concepts from this tutorial. If you get stuck, re-read the relevant chapter. **Good luck!**