iTarang Battery Dashboard

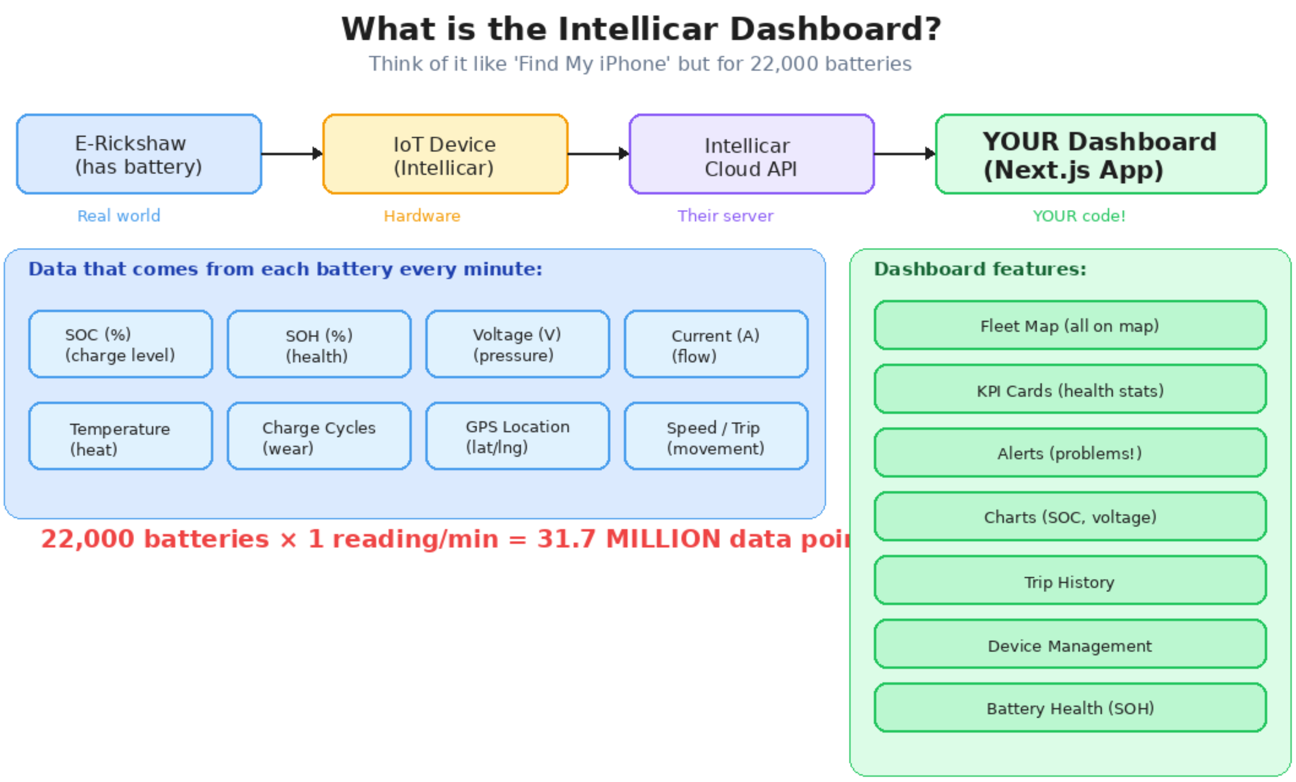
The Complete Beginner's Guide

Every concept in YOUR code, explained for a high school student

GitHub	github.com/apoorvgupta0709/intellicardashboard
Stack	Next.js 16 + React 18 + TypeScript + Tailwind CSS
Database	PostgreSQL + Drizzle ORM + Supabase + PostGIS
Viz	Recharts + Leaflet Maps + Heroicons + Tremor
Data	22,000 batteries × 31.7M data points/day
Parent App	iTarang CRM (leads, deals, loans, inventory)
New Scripts	ingest-historical.ts + ingest-live.ts (live daemon)

This tutorial includes **7 full-color diagrams** and **18 chapters** covering every technology in the project. Each code example is from YOUR actual codebase.

1 What Is This Project?

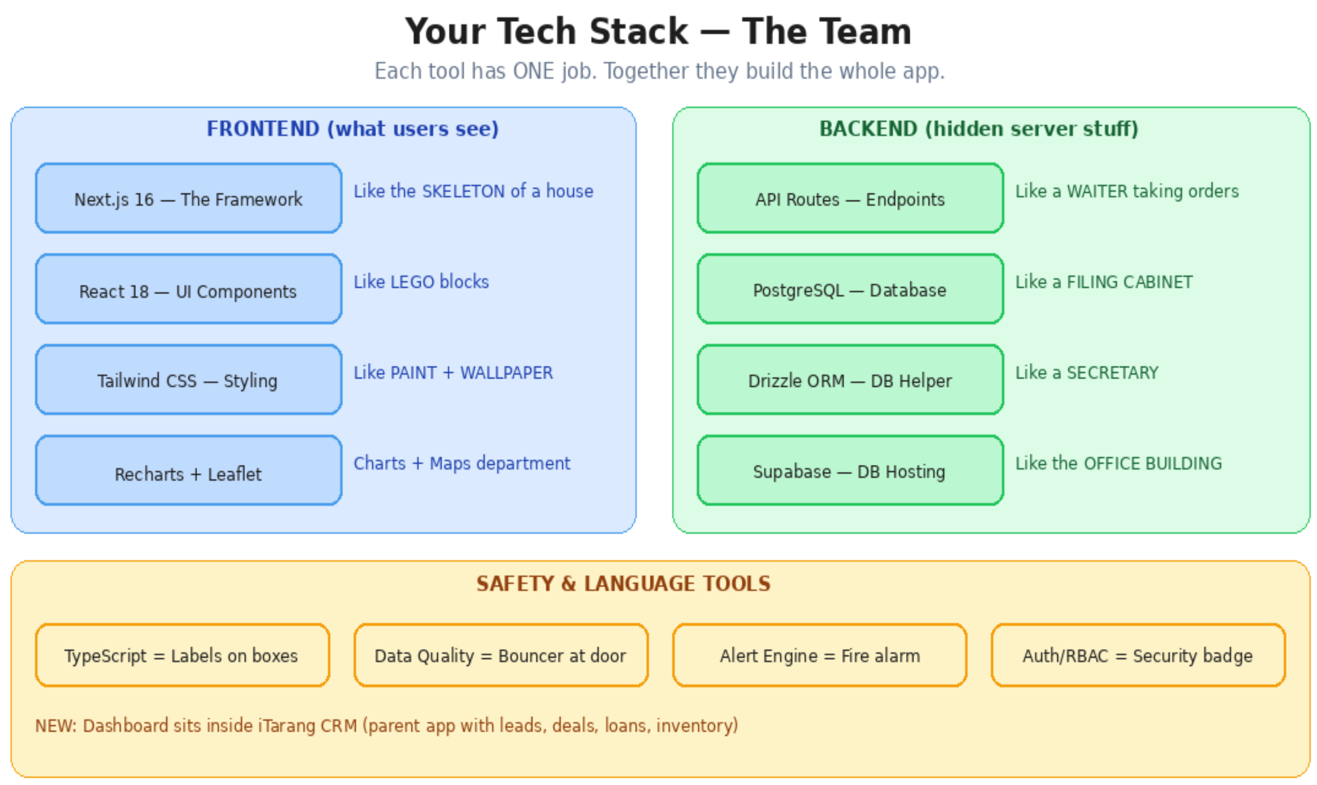


You sell lithium-ion batteries for e-rickshaws. 22,000+ are driving around India. Each has an IoT device (by Intellicar) that reports data **every 60 seconds**. Your dashboard is like **'Find My iPhone' for batteries**.

What You Want to Know	Battery Equivalent
Is it charged?	SOC (State of Charge) 0-100%
Is it healthy?	SOH (State of Health) 100%=new
Where is it?	GPS latitude + longitude
Is something wrong?	Alerts: overheating, dead, deep discharge
How much did it travel?	Trip history, distance, energy used

Scale: *22,000 batteries × 1 reading/min = 31.7 MILLION data points per day. That's like getting 31 million WhatsApp messages daily!*

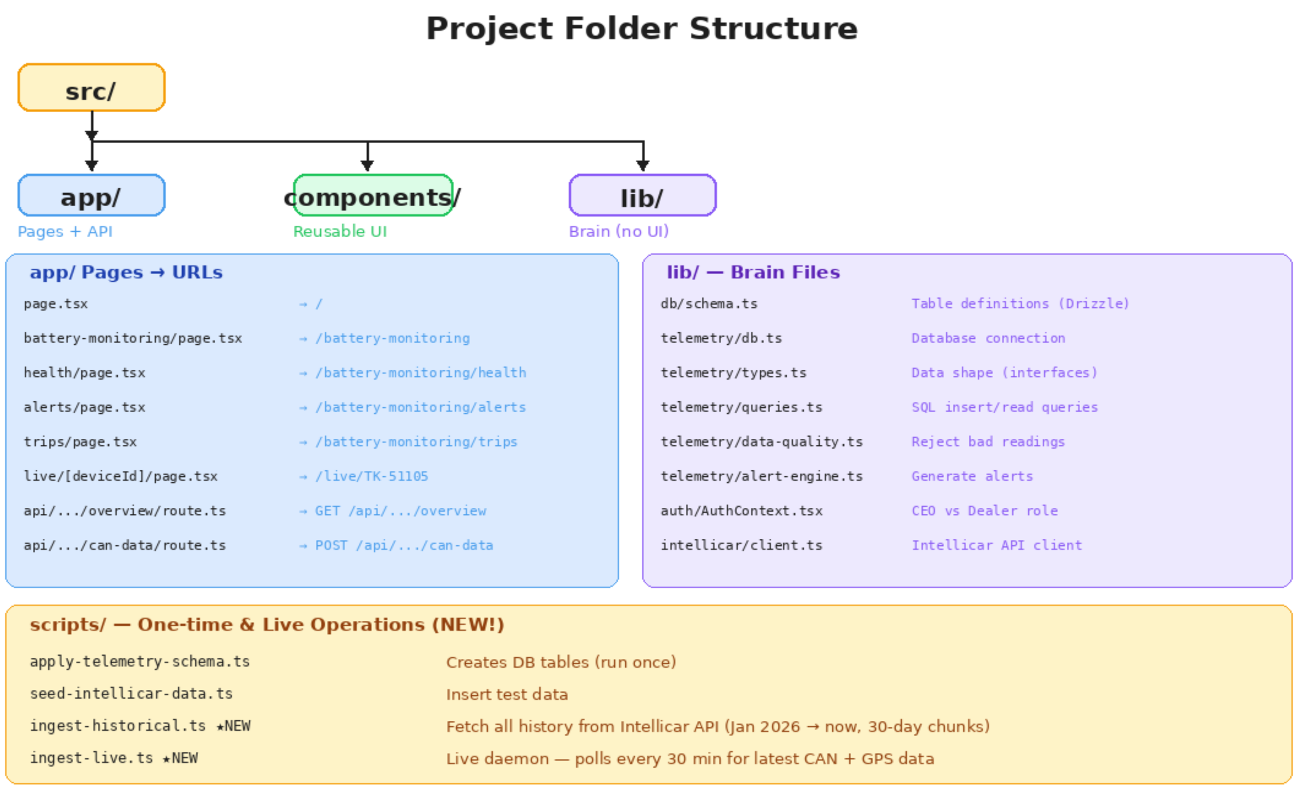
2 The Tech Stack


Your Tech Stack — The Team
Each tool has ONE job. Together they build the whole app.
FRONTEND (what users see)
Next.js 16 — The Framework
Like the SKELETON of a house
React 18 — UI Components
Like LEGO blocks
Tailwind CSS — Styling
Like PAINT + WALLPAPER
Recharts + Leaflet
Charts + Maps department
BACKEND (hidden server stuff)
API Routes — Endpoints
Like a WAITER taking orders
PostgreSQL — Database
Like a FILING CABINET
Drizzle ORM — DB Helper
Like a SECRETARY
Supabase — DB Hosting
Like the OFFICE BUILDING
SAFETY & LANGUAGE TOOLS
TypeScript = Labels on boxes
Data Quality = Bouncer at door
Alert Engine = Fire alarm
Auth/RBAC = Security badge
NEW: Dashboard sits inside iTarang CRM (parent app with leads, deals, loans, inventory)

Tool	Movie Analogy	What It Does
Next.js 16	Director	Controls pages, URLs, routing
React 18	Actors	Each button, card, chart is a React component
TypeScript	Script	Catches typos instantly (labels on everything)
Tailwind CSS	Costume Dept	Styling with class names like bg-blue-500
PostgreSQL	Film Archive	Stores all 31.7M daily data points
Drizzle ORM	Librarian	Type-safe database queries
Supabase	Studio	Hosts your PostgreSQL in the cloud
Recharts	Graphics	Bar charts, line charts, area charts
Leaflet	Map Dept	Battery locations on interactive India map
PostGIS	GPS Engine	Spatial queries on GPS coordinates

NEW: iTarang CRM is the parent app (leads, deals, loans, dealer portal, AI voice calls). The dashboard is a module inside this larger ecosystem.

3 File Structure



app/ Pages → URLs

page.tsx	→ /
battery-monitoring/page.tsx	→ /battery-monitoring
health/page.tsx	→ /battery-monitoring/health
alerts/page.tsx	→ /battery-monitoring/alerts
trips/page.tsx	→ /battery-monitoring/trips
live/[deviceId]/page.tsx	→ /live/TK-51105
api/.../overview/route.ts	→ GET /api/.../overview
api/.../can-data/route.ts	→ POST /api/.../can-data

lib/ — Brain Files

db/schema.ts	Table definitions (Drizzle)
telemetry/db.ts	Database connection
telemetry/types.ts	Data shape (interfaces)
telemetry/queries.ts	SQL insert/read queries
telemetry/data-quality.ts	Reject bad readings
telemetry/alert-engine.ts	Generate alerts
auth/AuthContext.tsx	CEO vs Dealer role
intellicar/client.ts	Intellicar API client

scripts/ — One-time & Live Operations (NEW!)

apply-telemetry-schema.ts	Creates DB tables (run once)
seed-intellicar-data.ts	Insert test data
ingest-historical.ts ★NEW	Fetch all history from Intellicar API (Jan 2026 → now, 30-day chunks)
ingest-live.ts ★NEW	Live daemon — polls every 30 min for latest CAN + GPS data

Key Naming Rules

- **(dashboard)** — Parentheses = group pages but DON'T add to URL. So URL is /battery-monitoring, not /dashboard/battery-monitoring.
- **[deviceId]** — Square brackets = variable URL. /live/TK-51105 and /live/TK-99999 both use the same page.tsx.
- **route.ts** = API endpoint (server-only). **page.tsx** = visible page.
- **layout.tsx** = shared wrapper (sidebar). Only the content area changes when navigating.

NEW Scripts Added

Script	What It Does	When To Run
ingest-historical.ts	Fetches ALL data from Intellicar API (Jan 2026→now) in 30-day chunks	Once (backfill)
ingest-live.ts	Daemon that polls every 30 min for latest CAN + GPS data per vehicle	Always running
apply-telemetry-schema.ts	Creates DB tables & indexes	Once (setup)
seed-intellicar-data.ts	Inserts test data	Once (dev)

4 TypeScript — Labels on Everything

JavaScript lets you put anything anywhere. TypeScript adds labels and catches mistakes **instantly**.

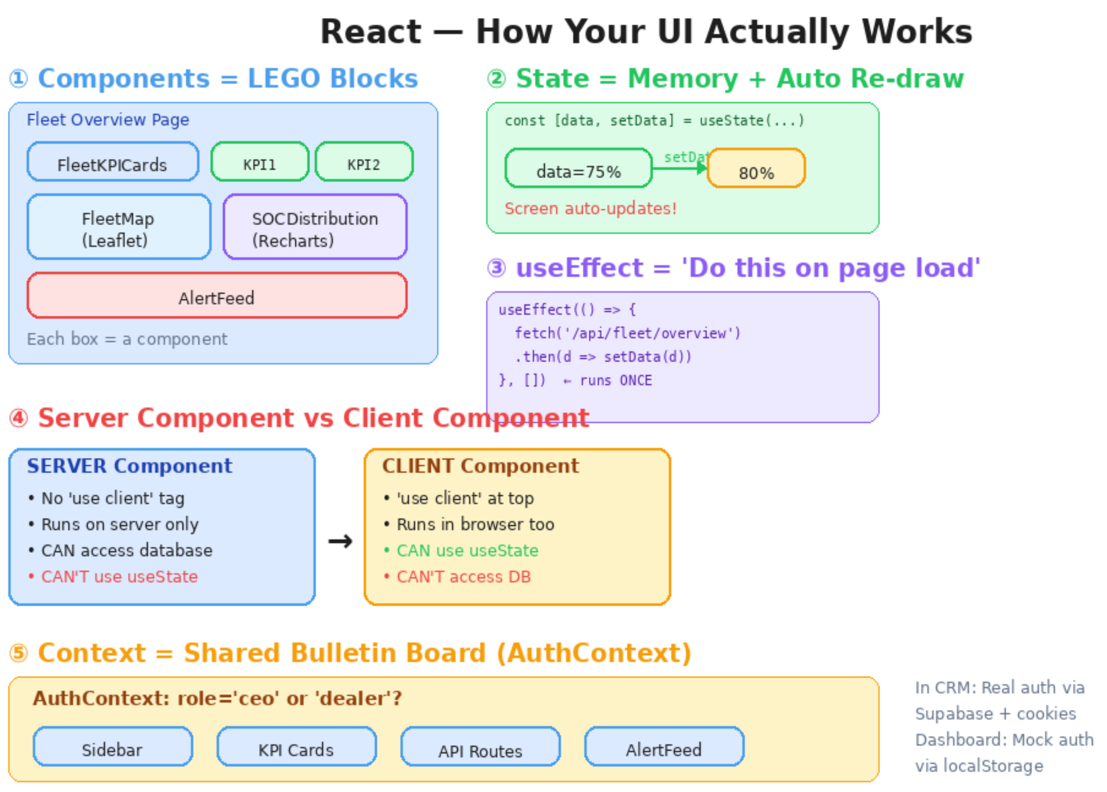
```
// JavaScript — chaos
let battery = { charge: 75 };
console.log(battery.chrage);  // Typo! JS says nothing. Bug found 3 weeks later.

// 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 (sensor might fail)
    soh: number | null;        // number|null = "or" — can be missing
    voltage: number | null;
    current: number | null;
    temperature: number | null;
    power_watts: number | null;
}
```

Syntax	Meaning	Example
number | null	"Either a number OR nothing"	soc: 75 or soc: null
string	Text value	"TK-51105"
?	Optional field	size?: number (can skip)
<T>	Generic (fill-in-the-blank)	postToIntellicar<any[]>(...)
export	Other files can import this	import { CANReading } from ...

5 React — Building the UI



```
const [data, setData] = useState(...)
```

```
useEffect(() => {
  fetch('/api/fleet/overview')
  .then(d => setData(d))
}, [])  ← runs ONCE
```

5 Core Concepts

1. Components = LEGO blocks. Each piece of UI is a function that returns JSX. BatteryGauge, FleetKPICards, AlertFeed are all components.

2. State = Memory that triggers re-draw. `useState` creates a variable where changing it (via setData) makes React automatically repaint the screen.

3. useEffect = 'When page loads, do this.' The empty `[]` means 'run once'. Put `[count]` to run when count changes.

4. Server vs Client Components. No keyword = server (fast, can access DB). 'use client' = browser (can use useState, onClick).

5. Context = Shared bulletin board. AuthContext lets any component know 'am I CEO or Dealer?' without passing props through 10 layers.

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 downloads)
Your files	page.tsx files	Most components/

6 Next.js — The Framework

URLs = Folder Names

```
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. When you click 'Health', **only the right panel changes**. The sidebar stays put — no flickering!

API Routes = Your Backend

```
// api/telemetry/fleet/overview/route.ts → 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(*) FROM device_battery_map WHERE is_active = TRUE
        ${auth.role === 'dealer' ? sql`AND dealer_id = ${auth.dealer_id}` : sql``}
    `);
    return NextResponse.json({ activeBatteries: Number(result[0]?.active_count) });
}
```

Dynamic Import = Load Heavy Stuff Lazily

```
// FleetMapDynamic.tsx — Leaflet needs browser's 'window' object
const FleetMap = dynamic(() => import('./FleetMap'), { ssr: false });
// ssr: false = "Skip server rendering. Only load in browser."
```

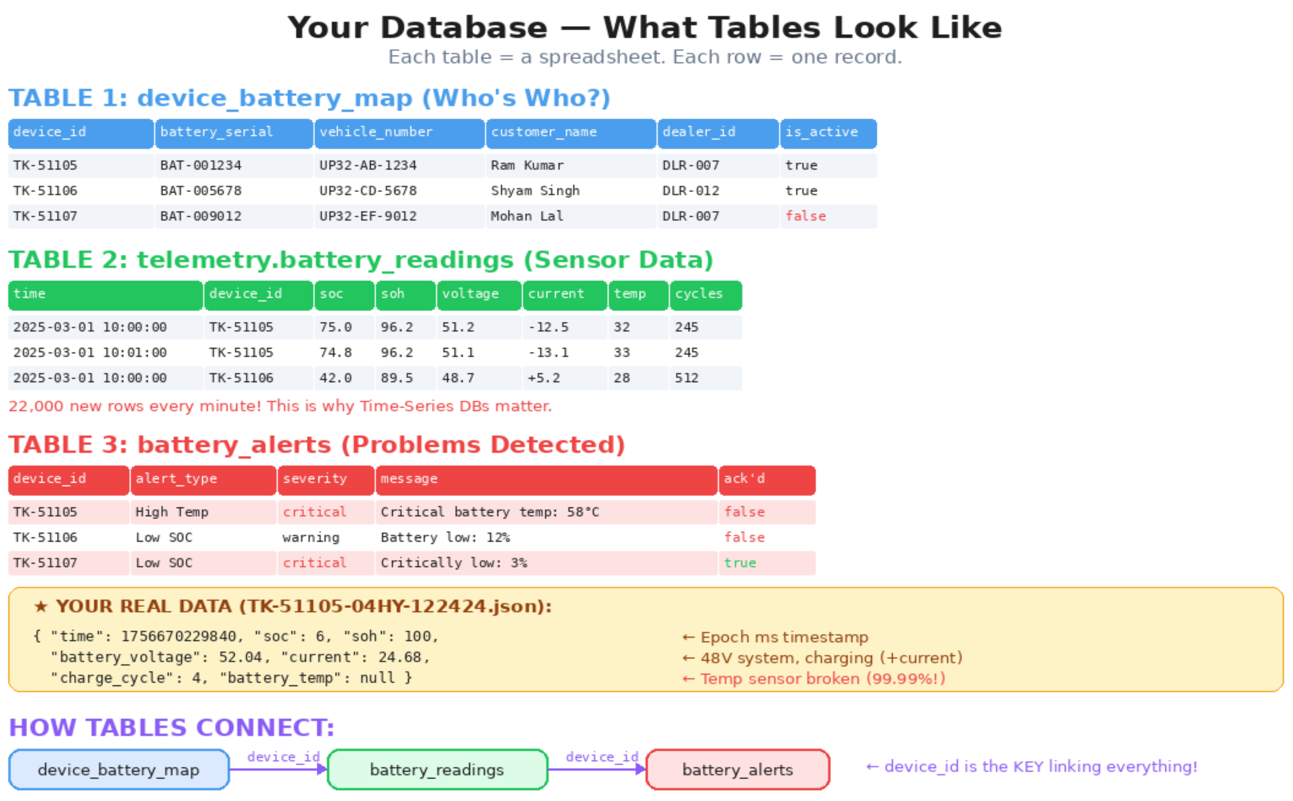
7 Tailwind CSS — Styling Without CSS Files

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

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

The **lg:** prefix = 'only on large screens.' Mobile stacks (1 col), desktop splits (3 cols). Responsive in one word!

8 The Database



Your Database — What Tables Look Like

Each table = a spreadsheet. Each row = one record.

TABLE 1: device_battery_map (Who's Who?)

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)

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

22,000 new rows every minute! This is why Time-Series DBs matter.

TABLE 3: battery_alerts (Problems Detected)

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

★ YOUR REAL DATA (TK-51105-04HY-122424.json):

```
{ "time": 1756670229840, "soc": 6, "soh": 100,          ← Epoch ms timestamp
  "battery_voltage": 52.04, "current": 24.68,           ← 48V system, charging (+current)
  "charge_cycle": 4, "battery_temp": null }             ← Temp sensor broken (99.99%!)
```

HOW TABLES CONNECT:

$$\begin{tikzcd} \text{device\_battery\_map} \arrow[r, "\text{device\_id}"] & \text{battery\_readings} \arrow[r, "\text{device\_id}"] & \text{battery\_alerts} \end{tikzcd}$$

← device_id is the KEY linking everything!

How Drizzle ORM Works

Your project uses **BOTH** approaches: Drizzle for CRM tables (type-safe), raw SQL for telemetry (TimescaleDB features).

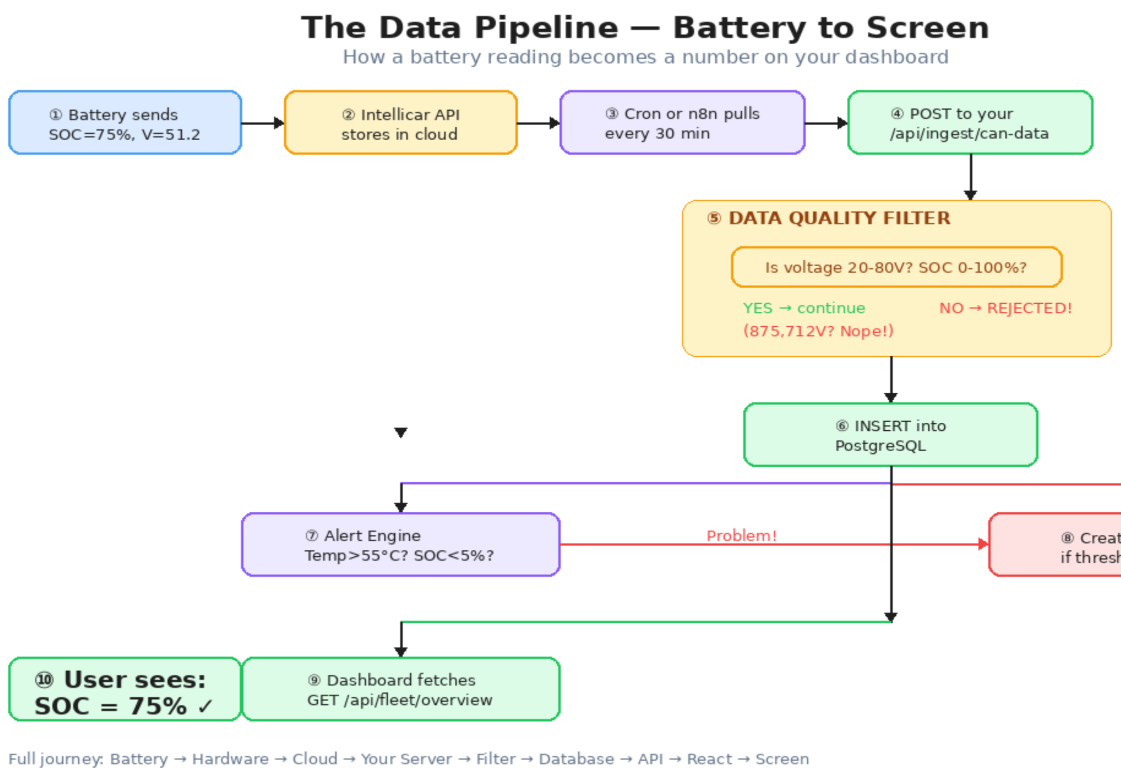
```
// Drizzle (safe, typed):
const devices = await db.select().from(deviceBatteryMap)
    .where(eq(deviceBatteryMap.dealer_id, 'DLR-007'));

// Raw SQL via Drizzle's sql tag (also safe — auto-parameterized):
const result = await telemetryDb.execute(sql`
    SELECT DISTINCT ON (device_id) device_id, soc, time
    FROM telemetry.battery_readings ORDER BY device_id, time DESC
`);
```

***SECURITY:** Drizzle's sql`` tag auto-escapes values to prevent SQL injection. NEVER build SQL with string concatenation like: 'SELECT * WHERE id=' + userId*

Real Data Format (from TK-51105-04HY-122424.json)

```
{ "time": 1756670229840,       // Epoch milliseconds
  "soc": 6,                     // 6% charge (very low!)
  "soh": 100,                   // Brand new battery
  "battery_voltage": 52.04,     // 48V system, ~full voltage
  "current": 24.68,             // Positive = CHARGING
  "charge_cycle": 4,            // Only 4 cycles (new)
  "battery_temp": null }         // Sensor broken (99.99% are null!)
```

9 The Data Pipeline


The Data Pipeline — Battery to Screen
How a battery reading becomes a number on your dashboard
① Battery sends SOC=75%, V=51.2
② Intellicar API stores in cloud
③ Cron or n8n pulls every 30 min
④ POST to your /api/ingest/can-data
⑤ DATA QUALITY FILTER
Is voltage 20-80V? SOC 0-100%?
YES → continue
NO → REJECTED!
(875,712V? Nope!)
⑥ INSERT into PostgreSQL
⑦ Alert Engine Temp>55°C? SOC<5%?
Problem!
⑧ Create ALERT if threshold broken
⑨ Dashboard fetches GET /api/fleet/overview
⑩ User sees: SOC = 75% ✓
Full journey: Battery → Hardware → Cloud → Your Server → Filter → Database → API → React → Screen

Step 1: Intellicar API Client (Token Caching)

Get token once, reuse for 55 min. Like getting a student ID instead of re-applying every day.

Step 2: Data Quality Filter

Bad Reading	Why Bad	Action
SOC = 123.71%	Can't be >100%	REJECTED
Voltage = 875,712V	48V system!	REJECTED
Current = 1,396,060A	More than lightning	REJECTED
Temp = null	Sensor broken (common)	Allowed (null=unknown)

Step 3: Alert Engine

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

NEW: Historical + Live Ingestion Scripts

```
// ingest-historical.ts — Backfill ALL data (Jan 2026 → now)
// Loops through all vehicles, fetches in 30-day chunks
// Inserts CAN readings + GPS with PostGIS ST_MakePoint
// Uses ON CONFLICT DO NOTHING to skip duplicates

// ingest-live.ts — Daemon (runs forever, polls every 30 min)
// Calls getlastgpsstatus + getlatestcan per vehicle
// Extracts CAN data from arbitrary key names (soc/SOC/battery_soc)
// Uses setInterval(fetchAndSaveLiveCycle, 30 * 60 * 1000)
```

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**.

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 = solve 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
```

Pattern 4: ON CONFLICT DO NOTHING — Idempotent inserts (NEW)

```
await db.insert(batteryReadings).values(chunk).onConflictDoNothing().execute();
// If (time, device_id) already exists, skip — no error, no duplicate
```

Used in both ingest scripts. Run them twice? No problem — duplicates are silently skipped.

Pattern 5: PostGIS — Spatial data

```
ST_SetSRID(ST_MakePoint(longitude, latitude), 4326)::geography
-- Creates a GPS point in the standard coordinate system (WGS 84)
```

Used in GPS ingestion. Enables 'find batteries within 5km of this dealer' queries.

11 Recharts — Drawing Charts

```
<ResponsiveContainer width="100%" height="100%">
  <BarChart data={data}>
    <XAxis dataKey="range" />     {/* "0-20%", "21-40%"... */}
    <YAxis />                      {/* Auto-calculated */}
    <Tooltip />                    {/* Hover info */}
    <Bar dataKey="count">          {/* Height = count value */}
      {data.map((_, i) => <Cell key={i} fill={COLORS[i]} />)}
    </Bar>
  </BarChart>
</ResponsiveContainer>
```

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

12 Leaflet — Interactive Maps

```
// FleetMapDynamic.tsx loads FleetMap.tsx ONLY in browser (ssr: false)
<MapContainer center={[26.8, 80.9]} zoom={6}>
  <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
  {devices.map(d => (
    <Marker position={[d.lat, d.lng]}>
      <Popup>{d.name} — SOC: {d.soc}%</Popup>
    </Marker>
  ))}
</MapContainer>
```

Two files because Leaflet needs browser's 'window'. dynamic({ ssr: false }) prevents server crash.

13 Authentication — Who Sees What

Role	What They See	Analogy
CEO	ALL batteries, ALL dealers, ALL alerts	Principal sees all students
Dealer	Only THEIR batteries and customers	Teacher sees only their class

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

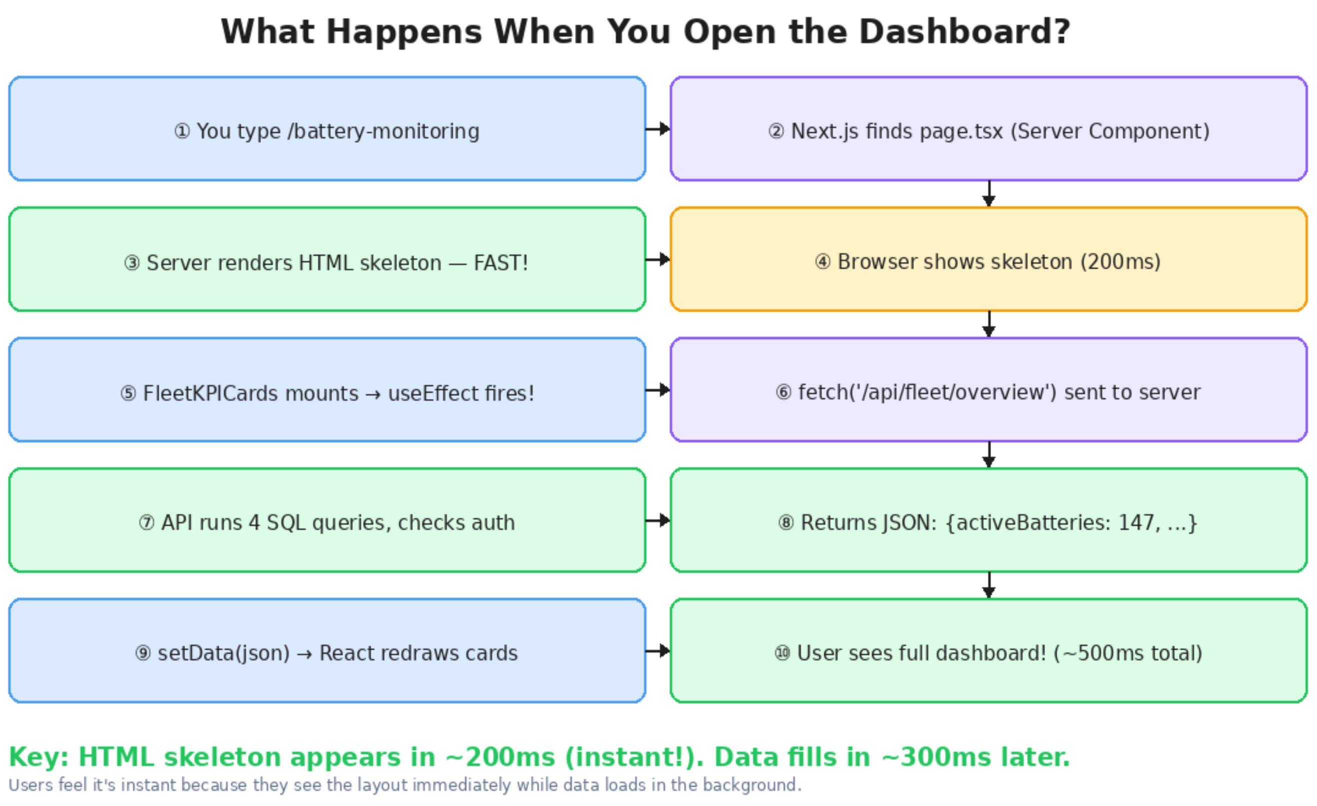
CRM has real auth via Supabase + cookies + middleware.ts. Dashboard uses mock auth (localStorage) for now.

14 Environment Variables — Secrets

Prefix	Who Sees	Example
No prefix	Server only (SAFE!)	DATABASE_URL, INTELLICAR_PASSWORD
NEXT_PUBLIC_	EVERYONE (browsers!)	NEXT_PUBLIC_MAP_URL

***NEVER** put NEXT_PUBLIC_ on database URLs or API keys. Browsers download that JavaScript — hackers can read it!*

15 The Full Request Lifecycle


What Happens When You Open the Dashboard?
① You type /battery-monitoring
② Next.js finds page.tsx (Server Component)
③ Server renders HTML skeleton — FAST!
④ Browser shows skeleton (200ms)
⑤ FleetKPICards mounts → useEffect fires!
⑥ fetch('/api/fleet/overview') sent to server
⑦ API runs 4 SQL queries, checks auth
⑧ Returns JSON: {activeBatteries: 147, ...}
⑨ setData(json) → React redraws cards
⑩ User sees full dashboard! (~500ms total)
Key: HTML skeleton appears in ~200ms (instant!). Data fills in ~300ms later.
Users feel it's instant because they see the layout immediately while data loads in the background.

Step	What	Where	Time
1	You type /battery-monitoring	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/fleet/overview)	Network	~215ms
7	API runs 4 SQL queries + auth check	Server+DB	~300ms
8	JSON returns: {activeBatteries:147}	Network	~400ms
9	setData(json) → React redraws	Browser	~410ms
10	User sees full dashboard!	Screen	~500ms

Key insight: HTML skeleton appears in ~200ms (instant!). Data fills in ~300ms later. Users feel it's instant because 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)	(dashboard) folder	Group without changing URL
[brackets]	[deviceId] folder	Variable URL segment
sql`` tag	All API routes	Safe SQL (prevents injection)
DISTINCT ON	Fleet overview	Latest reading per device
WITH...AS (CTE)	SOH averaging	Multi-step SQL
Conditional sql	Role queries	CEO vs Dealer filtering
ON CONFLICT DO NOTHING	Ingest scripts	Skip duplicates safely
ST_MakePoint	GPS ingestion	Create PostGIS point
Token caching	intellicar/client.ts	Reuse API tokens
Data quality filter	data-quality.ts	Reject 875,712V readings
Alert deduplication	alert-engine.ts	No duplicate alerts
30-day chunks	ingest-historical.ts	Avoid API timeouts
setInterval daemon	ingest-live.ts	Poll every 30 min forever

17 Glossary

Term	Meaning	Analogy
SOC	State of Charge (0-100%)	Phone battery %
SOH	State of Health (100%=new)	How worn out it is
CAN	Controller Area Network	Language batteries speak
API	Application Programming Interface	Waiter taking orders
JSON	JavaScript Object Notation	{ "name": "Ram" }
ORM	Object-Relational Mapping	Translator: code→SQL
SSR	Server-Side Rendering	Draw page on server first
Hydration	Activating server HTML	Waking up static page
CTE	Common Table Expression	Named temp result in SQL
RBAC	Role-Based Access Control	Different views per role
PostGIS	Spatial extension for Postgres	GPS math in database
Epoch	Unix timestamp	Milliseconds since 1970
Daemon	Always-running background process	Like a night watchman
Idempotent	Run twice = same result	Safe to re-run

18 Practice Challenges

Challenge 1: Add 'Avg Voltage' KPI Card

Difficulty: Easy **Uses: Ch 5, 6, 8**

Add a SQL query in fleet/overview route.ts. Add entry to kpiConfig in FleetKPICards.tsx.

Challenge 2: Make SOCDistribution fetch real data

Difficulty: Medium **Uses: Ch 5, 6, 10**

Replace mockData. Create /api/telemetry/analytics/soc-distribution with GROUP BY SOC range.

Challenge 3: Add search to Devices page

Difficulty: Medium **Uses: Ch 5, 6, 7**

Text input filtering by vehicle number or customer name. Use URL params + use-debounce.

Challenge 4: Add 'Rapid SOH Drop' alert

Difficulty: Hard **Uses: Ch 9, 10**

In alert-engine.ts: if SOH drops >5% in 30 days, create critical alert. Compare current vs 30-day-old reading.

Challenge 5: Add CSV export to Trips

Difficulty: Hard **Uses: Ch 5, 6**

Download button that fetches trip data and triggers browser .csv download using Blob API.

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