Axion – OTA and Telemetry

Axion – FINALIZED TECH STACK (Locked)

This stack is **industry-aligned, defensible in interviews, and executable by March**.

1️⃣ Backend & Core Platform

Primary Language: Java 17
Framework: Spring Boot 3.x

Why this is optimal

- Java + Spring = credibility for large-scale backend systems
- Spring Boot 3 + Java 17 shows modernity (not legacy)

Modules

- Spring WebFlux → telemetry ingestion (non-blocking)
- Spring MVC → admin, OTA, analytics APIs
- Spring Security → RBAC (Admin / Operator)
- Hibernate + JPA → metadata, users, OTA campaigns

2️⃣ Streaming & Messaging

Apache Kafka (non-negotiable)

Usage

- Telemetry ingestion topics
- Alert & critical-event topics
- OTA event streams
- Replay + DLQ

Data Format

- Protobuf (preferred) or Avro
- Schema Registry (even if mocked)

Why
This alone upgrades Axion from *college project → real system*.

3️⃣ Digital Twin Layer

Redis

Responsibilities

- Real-time vehicle state
- Online/offline status
- Last-known telemetry
- Twin lifecycle state (Healthy → Degraded → Critical)

Optional (nice-to-have):

- Redis Streams for state transitions

4️⃣ Storage

TimescaleDB

- All telemetry history
- Trend & anomaly queries

PostgreSQL

- Vehicles
- OTA campaigns
- Users & roles
- Audit logs

This separation is architecturally clean and interview-safe.

5️⃣ Analytics & ML (Major Scope)

Python

Libraries

- Pandas, NumPy
- Scikit-learn
- XGBoost (battery degradation)
- Optional: Prophet / LSTM (mentionable, not mandatory)

Serving

- FastAPI ML service
- REST-based inference

Key Principle
ML is **advisory**, not authoritative → confidence scores included.

6️⃣ OTA Orchestration

Spring Boot Service

Features

- Canary rollout
- Health-gated progression
- Auto rollback
- Signature verification (Uptane-inspired)

No real firmware flashing — simulation only (correct academic decision).

7️⃣ Frontend (FINAL)

React + TypeScript
Vite
Tailwind CSS

Charts

- Recharts / ECharts

Realtime

- WebSockets (STOMP or native WS)

This is already correct. No framework churn needed.

8️⃣ Observability (Major)

- Prometheus
- Grafana
- OpenTelemetry
- JSON structured logs

Even partial implementation is impressive.

9️⃣ Infra (Optional / Bonus)

- Docker
- Docker Compose
- Kubernetes (kind / minikube)
- GitHub Actions CI

Axion – Vendor-Neutral EV Fleet Digital Twin & OTA Orchestration Platform

🔍 Detailed Feature, Tech, and System Design Summary

1️⃣ Backend System – Feature Breakdown (Tech-Wise)

Axion follows a **microservice-oriented, event-driven architecture**.

1.1 Telemetry Ingestion Service

Responsibilities

- Accept telemetry from EV simulators / vendors
- Support **multiple protocols**
- Validate, normalize, and publish events

Features

- REST (HTTP) ingestion
- MQTT ingestion (IoT-style)
- Vendor-specific payload handling
- Retry & reconnect support
- Telemetry quality scoring (packet loss, delay, missing fields)

Tech Stack

- **Java 17**
- **Spring Boot**
- **Spring WebFlux** (non-blocking ingestion)
- Validation via custom schema validators
- Protobuf / JSON parsing

Output

- Publishes telemetry events to Kafka

1.2 Vendor-Neutral Data Normalization

Problem Solved

Different OEMs expose different telemetry formats.

Features

- Vendor adapters (Tesla-like, Tata-like, generic)

- Converts raw telemetry into a **canonical signal model**
    - speed
    - battery %
    - temperature
    - timestamp
    - vehicle_id

Inspiration

- COVESA Vehicle Signal Specification (conceptual)

Tech

- Java
- Adapter + Strategy pattern
- Kafka producer

2️⃣ Real-Time Event Pipeline (Core Backbone)

2.1 Kafka-Based Event Streaming

Features

- Decouples ingestion from processing
- Enables replay, scalability, and fault isolation
- Supports multiple downstream consumers

Topics

- telemetry.normal
- telemetry.alerts
- telemetry.critical
- ota.events
- telemetry.dlq

Tech

- **Apache Kafka**
- Kafka Streams (aggregation & filtering)

2.2 Fault Tolerance & Backpressure

Features

- Dead Letter Queue (DLQ) for invalid telemetry
- Graceful degradation under load
- Replay after fixes

3️⃣ Digital Twin Engine (Axion's Core Differentiator)

3.1 Live Vehicle Digital Twin

Concept

Each vehicle has an **in-memory digital twin** representing its latest state.

Stored State

- Latest telemetry values
- Online / offline status
- Health score
- OTA eligibility
- Last update timestamp

Tech

- **Redis**
- Versioned keys
- TTL for stale state

3.2 Offline State Retention

Features

- Retains last known state on disconnect
- Prevents data loss
- Auto-sync on reconnection

3.3 State Consistency Rules

Features

- Timestamp-based conflict resolution
- Prevents stale telemetry overwrites

- Enforced at twin update level

3.4 Twin Lifecycle State Machine

States

- HEALTHY
- DEGRADED
- CRITICAL
- RECOVERY

Transitions

Triggered by:

- Rule engine
- ML predictions
- OTA outcomes

🗄️ Persistent Storage & Query Layer

4.1 Time-Series Telemetry Storage

Features

- Historical telemetry storage
- Optimized for trends & anomaly detection

Tech

- **TimescaleDB**
- Hypertables
- Time-based partitioning

4.2 Fleet Metadata Storage

Stored Data

- Vehicle registry
- OTA campaigns
- Users & roles
- Audit logs

Tech

- **PostgreSQL**
- JPA / Hibernate

4.3 Fleet Query APIs

APIs

- List vehicles by health
- Find inactive vehicles
- Fetch telemetry history
- Segment fleet dynamically

5️⃣ Fleet Intelligence & Health Scoring (AI-Ready)

5.1 Rule-Based Health Scoring (Minor Scope)

Features

- Health score (0–100)
- Factors:
    - Battery behavior
    - Temperature anomalies
    - Connectivity stability

Tech

- **Python analytics service**
- Pandas + NumPy
- FastAPI

5.2 Explainable Insights

Features

- Human-readable explanations:
    - “Frequent high-temperature events”
    - “Rapid battery degradation detected”
- Transparent scoring logic

5.3 ML-Based Predictive Analytics (Major Scope)

ML Models

- Battery degradation prediction
- Failure probability estimation

Tech

- Scikit-learn
- XGBoost
- Optional LSTM (mentionable)

Output

- Prediction + confidence interval
- Integrated into digital twin

6️⃣ OTA Update Orchestration

6.1 OTA Simulation Engine

Features

- Simulated firmware updates
- Success / failure acknowledgments
- Safe for academic environments

Tech

- Spring Boot service
- Event-driven OTA state handling

6.2 Canary Deployment Strategy

Flow

- Rollout to small subset
- Monitor health + failures
- Expand gradually

6.3 Automated Rollback

Triggered When

- Failure rate exceeds threshold
- Health score drops post-update

6.4 Secure Update Validation

Features

- Signature verification
- Prevents unauthorized updates

Inspiration

- Uptane (conceptual, not full spec)

7️⃣ Analytics & Visualization Layer

7.1 Fleet-Level Analytics

Features

- Online vs offline distribution
- Health breakdown
- OTA success rates

7.2 Vehicle-Level Analytics

Features

- Telemetry trends
- Health score over time
- Event timeline

7.3 Root Cause Analysis (RCA)

Features

- Correlates:
    - Telemetry
    - OTA events
    - Health transitions

8️⃣ Security & Access Control

8.1 Authentication & Authorization

Features

- Role-based access:
    - Admin
    - Operator

Tech

- Spring Security
- JWT-based auth

8.2 Audit & Traceability

Logged Events

- OTA triggers
- Rollbacks
- Admin actions

9️⃣ Simulator Framework

9.1 EV Simulator

Features

- Python-based vehicle simulators
- Configurable vendor profiles
- Fault injection (battery drain, temp spike)

Tech

- Python
- Async simulation loops

🔟 Frontend System

Tech Stack

- **React + TypeScript**
- Vite
- Tailwind CSS
- WebSockets (live telemetry)
- Recharts / ECharts

UI Features

- Fleet overview dashboard
- Vehicle list with health indicators
- Digital twin visualization
- OTA campaign manager
- Analytics & insights pages

🔁 **End-to-End System Flow (High-Level)**

EV Simulator

↓

Telemetry Ingestion (REST / MQTT)

↓

Normalization Layer

↓

Kafka Topics

↓

Digital Twin Engine (Redis)

↓

├– TimescaleDB (History)

├– Health Scoring (Python ML)

├– OTA Orchestration

↓

Frontend Dashboard (React)

Axion – Simple Feature Summary (Team-Friendly)

Axion is a system that **monitors electric vehicles, understands their health, and safely manages software updates** — even when vehicles go offline or send messy data.

1️⃣ Vehicle Connectivity & Telemetry (Data Coming In)

What it does

- Vehicles send data like:
    - battery %
    - speed
    - temperature
- Vehicles can send data in **different formats**

What Axion handles

- Accepts data through:
    - HTTP (REST)
    - MQTT
- Works even if:
    - network is unstable
    - vehicle disconnects and reconnects

Why this is important

Different EV companies send data differently.
Axion adapts instead of forcing one format.

2️⃣ Data Standardization (Making Sense of Data)

What it does

- Converts all incoming vehicle data into **one common format**
- Example:
    - Tesla-style data
    - Tata-style data
      → both become the same internal structure

Why this matters

Once data is standard:

- analytics works for all vehicles

- UI stays simple
- backend logic is reusable

3️⃣ Real-Time Data Flow (System Backbone)

What it does

- All vehicle data goes through a central event system
- Different parts of Axion consume data independently

Why this matters

- System doesn't break if one part fails
- Data can be replayed if something goes wrong
- System scales easily

4️⃣ Digital Twin (Live Vehicle Representation)

What it is

For every vehicle, Axion keeps a **live digital copy**.

What the digital twin stores

- latest battery %
- latest temperature
- online/offline status
- last update time
- current health score

Even if a vehicle goes offline

- Last known data is saved
- State updates automatically when it reconnects

Why this matters

Fleet systems **cannot depend on perfect connectivity**.

5️⃣ Vehicle Health Scoring (Understanding Vehicle Condition)

What it does

- Gives every vehicle a health score (0–100)
- Score is based on:

    - battery behavior
    - temperature issues
    - connection stability

Important part

Axion also explains **why** a vehicle has that score:

- "High temperature events detected"
- "Battery draining faster than normal"

Why this matters

Numbers alone are useless without explanation.

6️⃣ Predictive Intelligence (Major Feature)

What it does

- Uses past data to predict:
    - battery degradation
    - possible failures
- Shows confidence with predictions

Why this matters

Instead of reacting after failure, Axion helps act **before** it happens.

7️⃣ OTA Update Management (Software Updates)

What it does

- Simulates software updates for vehicles
- Tracks:
    - success
    - failure
    - health impact

Safety features

- Updates go to a small group first (canary)
- If issues appear:
    - update is stopped
    - system rolls back automatically

Why this matters

A bad update should never break the entire fleet.

8️⃣ Analytics & Insights (Seeing the Big Picture)

Fleet-level view

- How many vehicles are:
  - healthy
  - warning
  - critical
- Online vs offline vehicles
- Update success rate

Vehicle-level view

- Health trend over time
- Battery degradation trend
- Event timeline

9️⃣ Security & Access Control

What it does

- Only authorized users can:
  - trigger updates
  - modify fleet settings
- All important actions are logged

Why this matters

Fleet systems are sensitive — mistakes must be traceable.

🔟 Simulator (Testing Without Real Vehicles)

What it does

- Simulates vehicles sending data
- Can create:
  - battery drain
  - temperature spikes

- network failures

Why this matters

We can test the system safely without real EVs.

🔁 **Simple System Flow (Explain Like This)**

1. Vehicles send data
2. Axion cleans and standardizes the data
3. Data updates the vehicle's digital twin
4. Health score and analytics are calculated
5. OTA updates are managed safely
6. UI shows live fleet status and insights

Axion Project Split: Minor vs Major

🟢 **MINOR PROJECT (Foundation + Core System)**

Focus: Real-time monitoring, digital twin, and safe OTA simulation

🎯 **Goal of Minor**

Build a **working EV fleet monitoring system** that:

- collects vehicle data
- shows live vehicle state
- calculates health
- safely simulates OTA updates

This alone is a **complete, usable system**.

✅ **Minor Project Features**

1️⃣ **Vehicle Telemetry & Connectivity**

- Vehicles send data (battery, temp, speed)
- Supports:
  - REST (HTTP)
  - MQTT
- Handles reconnects and intermittent network

📌 *Outcome:* System reliably receives vehicle data.

2️⃣ Data Normalization

- Convert different vehicle data formats into one common format
- Standard fields:
  - vehicle ID
  - battery %
  - temperature
  - timestamp

📌 *Outcome:* Backend and UI work the same for all vehicles.

3️⃣ Real-Time Data Pipeline

- All telemetry goes through Kafka
- Decouples ingestion from processing
- Supports replay and fault isolation

📌 *Outcome:* System scales and doesn't break easily.

4️⃣ Digital Twin (Core of Minor)

- One live digital twin per vehicle
- Stores:
  - latest telemetry
  - online/offline status
  - last update time
- Retains last known state when vehicle goes offline

📌 *Outcome:* Fleet can be monitored in real time.

5️⃣ Health Scoring (Rule-Based)

- Health score (0–100) per vehicle
- Based on:
  - battery behavior
  - temperature issues

    - connectivity stability
- Simple explanations included

📌 *Outcome:* Operators understand vehicle condition easily.

6️⃣ OTA Update Simulation (Basic)

- Simulated software updates
- Vehicles respond with success/failure
- No real firmware flashing

📌 *Outcome:* Safe OTA concept demonstration.

7️⃣ Analytics Dashboard (Basic)

- Fleet overview:
    - healthy vs unhealthy vehicles
    - online vs offline
- Vehicle-level telemetry charts

📌 *Outcome:* Clear visibility into fleet status.

8️⃣ Simulator Framework

- Python-based EV simulators
- Can simulate:
    - normal driving
    - battery drain
    - temperature spikes

📌 *Outcome:* System can be tested without real EVs.

🔒 Minor Project Freeze

At the end of Minor:

- System is stable
- Core architecture is fixed
- Data models and APIs are locked

This becomes the **base for Major**.

🔵 MAJOR PROJECT (Intelligence + Automation + Scale)

Focus: Prediction, decision-making, and enterprise-grade controls

🎯 Goal of Major

Upgrade Axion from:

"Monitoring system"
to
"Intelligent fleet management & orchestration platform"

🚀 Major Project Features

1️⃣ ML-Based Predictive Analytics

- Predict:
  - battery degradation
  - failure probability
- Confidence score for predictions

📌 *Improvement:* System warns **before** failure happens.

2️⃣ Advanced Health Intelligence

- Combine:
  - rule-based health
  - ML predictions
- Prescriptive suggestions:
  - "Exclude from OTA"
  - "Schedule inspection"

📌 *Improvement:* Health score becomes actionable.

3️⃣ Advanced OTA Orchestration

- Canary deployment
- Health-gated rollout

- Automated rollback on failure
- OTA approval workflow

📌 *Improvement:* OTA becomes safe, controlled, and realistic.

4️⃣ Digital Twin Intelligence

- Twin lifecycle states:
  - HEALTHY
  - DEGRADED
  - CRITICAL
  - RECOVERY
- Policy rules per vehicle:
  - battery limits
  - temperature limits
  - OTA eligibility rules

📌 *Improvement:* Digital twin makes decisions, not just stores data.

5️⃣ Root Cause Analysis (RCA)

- Correlates:
  - telemetry
  - OTA events
  - health changes
- Event timelines for failures

📌 *Improvement:* Easier debugging and post-mortems.

6️⃣ Security & Governance

- Role-based access (Admin / Operator)
- Audit logs for:
  - OTA triggers
  - rollbacks
  - admin actions

📌 *Improvement:* System becomes enterprise-ready.

7️⃣ Observability & Reliability

- Metrics, logs, dashboards
- System health monitoring
- Graceful degradation testing

📌 *Improvement:* You can observe and trust the system.

8️⃣ Advanced Analytics & Forecasting

- Fleet health forecasting
- Trend-based insights
- Risk indicators for OTA campaigns

📌 *Improvement:* Strategic, long-term decision support.

Axion – Canonical Telemetry Schema (MINOR FREEZE)

This is the **one internal language** every vehicle must speak.

1️⃣ Top-Level Envelope (Non-Negotiable)

Every telemetry message MUST look like this internally, regardless of REST / MQTT / vendor.

```
{
  "schema_version": "1.0",
  "vehicle_id": "EV-001",
  "vendor": "SIMULATED_TESLA",
  "timestamp": "2026-01-25T18:32:45Z",
  "ingestion_ts": "2026-01-25T18:32:47Z",
  "telemetry": { },
  "connection": { }
}
```

Why this envelope exists

- schema_version → Major-safe evolution
- timestamp ≠ ingestion_ts → handles delayed packets
- Clean separation of **vehicle data** vs **network reality**

2️⃣ Telemetry Payload (Vehicle State)

This is what the **digital twin actually cares about**.

"telemetry": {

"speed_kmph": 64.2,

"battery_soc_pct": 78.5,

"battery_temp_c": 34.1,

"motor_temp_c": 41.3,

"ambient_temp_c": 29.0,

"odometer_km": 18234.6

}

Rules (LOCK THESE)

- Units are **explicit** (kmph, pct, celsius)
- No nested vendor nonsense
- Flat structure → easy Kafka, easy Redis, easy ML

Optional-but-allowed

If value is unknown:

"battery_temp_c": null

❌ **Never omit keys** once schema is frozen.

3️⃣ Connection Metadata (Critical for Health)

This is NOT telemetry — this is **network truth**.

"connection": {

"protocol": "MQTT",

"signal_strength": -71,

"sequence_number": 18492,

"packet_loss_pct": 0.6,

"is_heartbeat": false

}

Why this matters

- Health scoring needs connectivity quality

- Digital twin needs sequence ordering
- Replay logic needs idempotency

4️⃣ Minimal Required Fields (Hard Validation)

A message is **rejected** if missing:

vehicle_id

timestamp

telemetry.battery_soc_pct

Everything else can degrade gracefully.

This avoids garbage-in poisoning your system.

5️⃣ Fields You Intentionally EXCLUDE (Important)

You do **NOT** include these in Minor:

❌ GPS / location
❌ Driver behavior
❌ Charging station metadata
❌ Firmware binary info
❌ VIN / personal identifiers

Why

- Not needed for core learning goals
- Explodes scope
- Raises privacy questions in evaluation

You can **mention them as future extensions**.

6️⃣ Digital Twin Mapping (1:1, No Magic)

Each incoming message updates this Redis structure:

{

"vehicle_id": "EV-001",

"last_seen": "2026-01-25T18:32:47Z",

"online": true,

"telemetry": { },

"health_score": 87,

"health_state": "HEALTHY"

}

Rules:

- If now - last_seen > X seconds → online = false
- Newer timestamp always wins
- Older packets are ignored (but logged)

7️⃣ Health Engine Dependency (Schema-Aware)

Your rule-based health engine ONLY reads:

battery_soc_pct

battery_temp_c

packet_loss_pct

last_seen

Anything outside this is **ignored in Minor**.

This is intentional containment.

8️⃣ Kafka Compatibility (Critical)

This schema:

- Is **append-friendly**
- Works with JSON or Protobuf
- Can be versioned safely

Example topic message:

{

"key": "EV-001",

"value": { canonical telemetry }

}

Keying by vehicle_id guarantees ordering per vehicle.

9️⃣ Schema Versioning Strategy (Major-Proof)

Minor

"schema_version": "1.0"

Major (example)

"schema_version": "1.1"

Rules:

- New fields → allowed
- Renaming fields → forbidden
- Removing fields → forbidden

Adapters handle backward compatibility.

🔒 **FINAL FREEZE DECLARATION (Say This Out Loud)**

"All ingestion formats are converted into a single canonical telemetry schema before entering Kafka. This schema is frozen at the end of Minor and treated as a contract for all downstream systems."

That sentence alone wins trust.

Minor Project Work Division (Axion – Minor Phase)

🧠 **Core Principle (Non-Negotiable)**

- **You** own **system correctness, architecture, and backend truth**
- **Kajol** owns **simulation, data realism, analytics, and validation**

No overlap. Clear handoffs. One throat to choke per domain.

👩‍💻 **Your Ownership — *Platform & Core System Lead***

You are the **system backbone owner**. If it runs, scales, or breaks — it's on you.

1️⃣ **Vehicle Telemetry & Connectivity**

You own:

- REST ingestion service (Spring Boot)
- MQTT ingestion service
- Connection lifecycle (connect / disconnect / heartbeat)
- Retry & intermittent network handling (idempotency, last-seen logic)

Deliverable:

Fleet can push data reliably under flaky conditions.

2️⃣ **Data Normalization Layer**

You own:

- Canonical telemetry schema
- Adapter pattern for REST vs MQTT payloads
- Validation rules (bad data rejection, defaults)

Deliverable:

Every vehicle speaks one language internally.

3️⃣ Real-Time Data Pipeline (Kafka)

You own:

- Kafka topics design
- Producer logic from ingestion layer
- Consumer services (digital twin updater, health scorer)
- Replay strategy & fault isolation

Deliverable:

Ingestion ≠ processing. System is resilient.

4️⃣ Digital Twin (Core of Minor)

You own:

- Digital twin data model
- In-memory + persistent state strategy
- Online/offline detection
- Last-known-state retention

Deliverable:

One authoritative live state per vehicle.

5️⃣ Health Scoring Engine (Rule-Based)

You own:

- Health scoring rules
- Scoring algorithm
- Explanation engine (why score dropped)
- API to fetch vehicle health

Deliverable:

Health score is explainable, not magic.

6️⃣ OTA Update Simulation (Backend Side)

You own:

- OTA job orchestration
- State machine (PENDING → IN_PROGRESS → SUCCESS/FAIL)
- Timeout & rollback simulation
- Kafka-based OTA events

Deliverable:

OTA logic is safe, deterministic, and demo-ready.

7️⃣ Backend APIs (Contract Owner)

You own:

- Fleet APIs
- Vehicle APIs
- OTA APIs
- Health APIs
- OpenAPI / Swagger as **locked contract**

Deliverable:

APIs freeze at Minor end. No churn.

👩‍🔬 Kajol's Ownership — *Simulation, Analytics & Validation Lead*

Kajol is your **reality injector**. If data is fake or insights are weak — it's on her.

8️⃣ Simulator Framework (Python)

Kajol owns:

- Python EV simulator architecture
- Multiple vehicle profiles
- Scenario engine:
    - Normal driving
    - Battery drain

    - Temp spikes
    - Network dropouts
- REST + MQTT emitters

Deliverable:

System behaves like it's talking to real EVs.

9️⃣ OTA Simulation (Vehicle Side)

Kajol owns:

- Simulator OTA client
- Randomized success/failure
- Delayed responses
- Fault injection (mid-update disconnect)

Deliverable:

OTA demo feels real, not scripted.

🔟 Analytics & Dashboard (Basic)

Kajol owns:

- Fleet-level aggregates
- Healthy vs unhealthy stats
- Online vs offline stats
- Vehicle telemetry charts
- Data queries (read-only)

Deliverable:

At-a-glance operational clarity.

1️⃣1️⃣ Data Validation & Edge Case Testing

Kajol owns:

- Stress testing with simulators
- Invalid payload testing
- Kafka lag & replay tests
- Health score sanity checks

Deliverable:

Minor freeze means *actually stable*, not “seems fine”.

1️⃣ 2️⃣ Documentation & Demo Scripts

Kajol owns:

- Simulator usage docs
- Demo flows (what to click, what to show)
- Test scenarios for evaluation day

Deliverable:

Anyone can run and understand the system.

🔒 Minor Freeze Rules (Be ruthless here)

At Minor completion:

- ❌ No schema changes
- ❌ No API changes
- ❌ No Kafka topic changes
- ❌ No core logic refactors

Only **additive features** allowed in Major.