Solution Architecture Blueprint: Field Execution & Monitoring MVP

1. Executive Architectural Vision

As the Principal Solutions Architect for this initiative, my vision is to establish a high-integrity bridge between the chaotic, low-connectivity environment of field operations and the rigorous data requirements of corporate intelligence. For this Sprint 1 MVP, I have engineered a multi-tier architecture that prioritizes immediate field utility without compromising the "Single Source of Truth." This approach is a strategic necessity; it allows us to decouple the mobile execution layer from the central repository, ensuring the system remains responsive to the Sales Rep while providing Business Unit (BU) managers with validated, auditable data.My primary architectural objectives are **Data Reliability** , **Offline Resilience** , and **Centralized Logic** . By adopting a "Mobile-First, Offline-Always" philosophy, I am mitigating the inherent risks of weak field connectivity. We treat the mobile device as an autonomous data collection engine. This ensures that a Rep's productivity is never tethered to a signal bar, while the "Central Brain" (the API) ensures that only clean, business-rule-compliant data enters our persistent storage. This architecture provides the scaffolding necessary for rapid Sprint 1 delivery while maintaining a clear path toward global scalability.

2. High-Level System Topography

The architecture follows a rigorous three-tier model (Mobile, API, Database) designed to decouple field operations from business intelligence. This separation is vital for maintaining a stateless service layer and a protected data core.

The "No Direct Access" Rule

A non-negotiable architectural constraint is the **No Direct Access** rule. Mobile and Web clients are strictly prohibited from bypassing the API to interact with the PostgreSQL database. This ensures that every transaction is intercepted by our service tier for business logic validation, role-based authorization, and audit logging. Direct database access would invite data corruption and bypass the security protocols I have established.

Technology Stack Mapping

Component,Technology,Functional Responsibility,Sprint 1 Priority
Mobile Application,Flutter,"Field data collection, offline persistence, and UI for Sales Reps.",High
Local Storage,Hive,"Strategic ""fast-track"" key-value storage for MVP velocity.",High
Service Tier (API),ASP.NET Core,"Stateless business logic, JWT Auth, and EF Core orchestration.",High
API Documentation,Swagger,Automated documentation and interactive endpoint testing.,High
Primary Database,PostgreSQL,"Relational ""Single Source of Truth"" for all master and transactional data.",High
Management Portal,React + Tailwind,Web-based monitoring and configuration for BU Managers.,Medium

Maps/Location,Google Maps Platform,Capture of lat/long for shop master data (Minimal for S1).,Medium

3. Mobile Tier: Offline Resilience with Hive

For the Sprint 1 MVP, I have selected **Hive** as the local storage engine. I have made a calculated decision to accept Hive as **strategic debt** to favor development velocity. While SQLite (via Drift) offers superior relational structure for complex querying, Hive's high-speed key-value "Box" system allows us to reach a demo-ready state faster. We will evaluate a migration to Drift in Sprint 2 if the relational complexity on the device increases.

The "Box" Concept

Data is partitioned into specialized Hive Boxes to mirror field entities:

- **Shops Box:** Locally cached metadata for assigned shops.
- **OSA Records Box:** Temporary storage for shelf-check results and stock availability.
- **Orders/Returns Box:** Drafts of sales transactions and SKU-level return reasons.

Bulk Sync Strategy

To preserve battery and ensure data consistency in low-signal areas, I have implemented a **Bulk Sync strategy (POST /sync/day)** . The mobile app functions as a localized silo throughout the day, aggregating all activity. At the end of the shift, the Rep triggers a single synchronization event. This simplifies error handling and provides a definitive reconciliation point.

Technical Visualization: Sync Sequence Flow

```
sequenceDiagram
    participant Rep as Sales Rep (Flutter)
    participant Hive as Local Storage (Hive)
    participant API as Central Brain (ASP.NET Core)
    participant DB as PostgreSQL

    Rep->>Hive: Commit Visit, OSA, and Order Data
    Note over Rep, Hive: Offline Operations (All Day)
    Rep->>API: POST /sync/day (Encrypted JSON Payload)
    API->>API: Validate JWT & Business Rules
    API->>DB: Atomic Transactional Commit (EF Core)
    DB-->>API: Success Response
    API-->>Rep: 200 OK / Clear Local Boxes
```

4. The Service Tier: ASP.NET Core as the "Central Brain"

The ASP.NET Core Web API is the "Central Brain" of this architecture. It is designed to be **stateless** , relying on JWTs for context rather than server-side sessions. I have utilized **Entity**

Framework Core (EF Core) to implement a **Code-First** approach, bridging our object-oriented logic with the relational schema.

The Visit Frequency Engine

The API manages the core execution logic using a dynamic frequency engine. The rule is defined as: lastVisitedDate + frequencyDays <= today Unlike hard-coded systems, frequencyDays is a dynamic variable retrieved from the Settings table, allowing BU Managers to adjust the execution tempo (defaulting to 14 days) via the React dashboard without a code deployment.

Core Functional Endpoints & The "So What?"

- **Auth (/auth/login):** Secures the perimeter and issues roles. **So what?** Ensures only authorized Reps can submit data.
- **Shops (/shops/due):** Executes the frequency engine logic. **So what?** Directs the Rep to high-priority targets, optimizing route efficiency.
- **Sync (/sync/day):** The architectural pivot point. **So what?** It triggers the transition from **unverified field data** to **auditable corporate records** , ensuring stock and sales figures are finalized.
- **Settings (/settings/visit-frequency):** Exposes system parameters. **So what?** Shifts control from IT to Business Units for operational agility.

5. Security Architecture: JWT and Role-Based Access Control (RBAC)

Identity management is central to the dual-user ecosystem. I have implemented a JWT-based flow where the token is stored in flutter_secure_storage on the mobile device, ensuring that sensitive credentials are never held in plain text.

RBAC Implementation

Permissions are strictly bifurcated:

- **Rep Role:** Authorized for field execution—accessing shop lists, starting visits, and syncing day-end data.
- **BU Role:** Authorized for monitoring and configuration—accessing territory-wide summaries and modifying global settings like visit frequency.The React Admin Dashboard consumes the same "Central Brain" API, ensuring that the same business rules and security constraints apply to the web view as they do to the mobile app.

6. Data Tier: PostgreSQL and Schema Integrity

PostgreSQL serves as our Single Source of Truth. The schema is designed to support high-performance monitoring queries and complex relational dependencies.

Relational Logic & Filtering

To support the BU Dashboard's requirement for territory filtering, I have established a hierarchical relationship: Users are assigned to Territories, and Shops are linked to those

Territories. This allows the API to serve the "Due Today" list specifically to the Rep assigned to that region while allowing BU Managers to view data aggregated by territory or individual Rep.

Critical Tables

- **day_sessions:** Tracks the lifecycle of a workday.
- **visit_osa:** Stores stock checks with specific **Reason Codes** : *No Stock, Backroom, Not Listed,* or *Competitor Blocked* .
- **lorry_stock_opening:** Captures initial inventory for reconciliation.
- **returns:** Tracks SKU-level returns with mandatory reason codes.

7. Operational Workflow: From Field Visit to Web Visibility

The architecture supports a seamless end-to-end lifecycle, ensuring that every action in the field results in an expected output in the data tier.

The Step-by-Step Workflow

- **Start of Day:** Rep selects vehicle and enters opening lorry stock.
- *Expected Output:* A day_session and lorry_stock_opening record in PostgreSQL.
- **Visit Execution:** Rep selects a shop from the "Due Today" list and records On-Shelf Availability (OSA). If an item is missing, they must select a reason code (e.g., *Competitor Blocked*).
- **Smart Suggestion Order:** To maximize sales, the system generates suggested orders based on:
- *Architectural Constraints:* The API applies **case pack rounding** , **Minimum Order Quantity (MOQ)** warnings, and **Max order** limits before persistence.
- **End of Day Reconciliation:** The Rep records closing stock and returns. The system executes the reconciliation logic:
- **Bulk Sync:** POST /sync/day sends the entire payload.
- *Expected Output:* A finalized audit trail visible in the BU Monitoring Dashboard.

8. Conclusion: Architecture Validation and Future Extensibility

This blueprint confirms that the Field Execution & Monitoring MVP is built on a robust, professional-grade foundation. By utilizing a "Central Brain" API to mediate between an offline-resilient Flutter front-end and a PostgreSQL backbone, we have ensured both developer velocity and data integrity.The system is ready for the Sprint 1 demo. Looking toward Sprint 2, the foundation for **Google Maps Platform** integration is already in place; the database schema currently supports latitude/longitude capture for shop master data, allowing for seamless transition into route optimization and geo-fencing. This architecture is not just a prototype—it is a scalable production-ready engine.