CONCEPT STUDY · INTERACTIVE PROTOTYPE

RouteBoard: Manifest Freshness & Resequence Diffing

Data freshness timestamps, dead-zone stale manifest warnings, side-by-side sequence diffing, and offline proof of delivery for urban delivery couriers.

INDUSTRY VERTICAL
Logistics & Fleet Transport

DEMONSTRATED FOCUS
Multi-Drop Last-Mile Delivery

SIMULATED PROTOTYPE STACK
Next.js / TypeScript / React

PROPOSED PRODUCTION RUNTIME
React Native / WatermelonDB

Executive Summary

Urban courier operations require real-time synchronization between dispatch supervisors and field couriers. Couriers operating in subterranean concrete freight bays and commercial parking basements face persistent cellular signal loss. Standard delivery apps fail to indicate that on-device manifests are stale, leading to expensive wrong turns and delayed rush deliveries.

RouteBoard surfaces explicit sync timestamps on every stop in the route manifest, triggering prominent amber warnings when data freshness lapses past acceptable thresholds. When dispatch issues priority re-sequencing, RouteBoard renders a clear side-by-side diff that highlights estimated mileage savings and empowers couriers to accept or override changes. All proof-of-delivery signatures and barcode scans persist to encrypted local storage.

Urban Fleet Manifests: Why Static Views Fail

Last-mile courier runs are highly dynamic. Emergency rush orders, building access gate updates, and customer cancellations occur continuously throughout a shift:

Common Dispatch Failure Modes:

1. The Silent Stale Data Trap: Drivers rely on cached stop manifests that are 30 minutes old, unaware that dispatch re-routed emergency medical packages, causing costly urban backtracking.

2. Disorienting Sequence Swaps: Applications that silently re-sort stop lists mid-shift confuse couriers, who find their next navigation destination suddenly altered without context.

3. Lost Offline POD Signatures: Recipient signatures captured in underground loading docks are discarded by fragile web views, forcing drivers to make embarrassing second visits.

RouteBoard makes manifest data freshness visible and intuitive at all times.

Architectural Blueprint: Manifest Freshness Engine

RouteBoard decouples the local on-device route view from server-side dispatch mutations:

```
// WatermelonDB Manifest Stop Model (TypeScript)
interface RouteStop {
  id: string;
  sequenceNumber: number;
  proposedSequenceNumber: number | null;
  lastSyncedAt: number; // Unix timestamp
  isStale: boolean;
  podSignatureBlob: string | null;
}
```

Key architectural properties:

- **Monotonic Freshness Timestamps:** Local SQLite records track the exact millisecond of the last server handshake, surfacing elapsed duration (“Synced 24m ago”).
- **Side-by-Side Manifest Diffing:** Incoming sequence mutations from dispatch are staged in a “proposed” state until explicitly accepted or merged by the courier.
- **Persistent Offline POD Storage:** Digital signatures and barcode scans are written to encrypted local storage with automatic queue replay on cellular reconnection.

User Experience Decisions: Non-Disruptive Resequencing

RouteBoard respects driver situational awareness while navigating dense urban streets:

1. Amber Stale Cache Alert

When cellular signal drops in an underground bay, a clear amber badge warns the courier that manifest updates may be pending.

2. Visual Route Diff Modal

Dispatch re-orderings are displayed as side-by-side route diffs, highlighting priority rush stops and estimated mileage savings.

3. Offline Signature Pad

Recipients sign directly on screen inside basement offices with full offline confirmation and barcode validation.

4. Automatic Dispatch Sync

Upon exiting the loading bay, the app automatically reconciles POD records with central dispatch in under 250 milliseconds.

Interface State Sequence: 5 Core Screens

Screen 01 · Route Manifest

Ordered 35-stop delivery schedule with freshness timestamps and package count summary.

`State: Route Manifest · 35 Stops`

Screen 02 · Stop Details

Gate codes (#8821), freight elevator notes, and recipient contact info for Stop #14.

`State: Stop Detail · Loading Dock`

Screen 03 · Stale Manifest Warning

Amber notification indicating 24-minute sync lapse and flagging dispatch priority stop updates.

`State: Stale Cache (24m) · Dead Zone`

Screen 04 · Resequence Diff

Side-by-side comparison showing original route order vs proposed dispatch rush order.

`State: Resequence Diff · Confirmed`

Edge Case Resilience & Logistics Guardrails

Resilience Matrix:

1. Driver Inaccessible Loading Gate: If an access code fails offline, RouteBoard queues an exception note with photo verification, allowing the courier to defer the stop without invalidating subsequent route points.

2. Dispatch Cancellation During Transit: If a package is cancelled by the shipper while the van is en route, the stop is flagged with an immediate audio cue and re-routed to depot return.

3. Barcode Scanner Hardware Failure: The app provides an accessible manual alphanumeric entry fallback with instant checksum validation.

Technical Specification: Prototype vs. Native Runtime

Simulated Web Prototype (Delivered):

Built with Next.js 15, React 19, and TypeScript. Implements an in-memory client state machine that simulates loading dock dead zones, stale manifest flags, side-by-side route diffs, and offline POD signature storage.

Target Production Runtime (Specification):

- **Mobile Client:** React Native / Expo with TypeScript.
- **Local Database:** WatermelonDB (SQLite) for reactive multi-stop manifests.
- **Navigation:** Mapbox SDK with turn-by-turn offline tile caching.
- **Telemetry:** Background GPS tracking with exponential backoff sync queue.

Observed Metrics, Target Thresholds & Handover

RESEQUENCE LATENCY

< 250ms

Local manifest reordering time upon sequence conflict.

OFFLINE POD CAPTURE

14 seconds

End-to-end signature and barcode capture without connection.

Engineering Handover Terms:

Upon engagement, FFNA delivers the complete React Native mobile codebase, Mapbox navigation adapters, dispatch WebSocket server architectures, and WatermelonDB sync schemas under full client intellectual property assignment.