ElectricSQL Local-First Sync Strategy

Complete Technical Report for app_barcode Module

AIMI Engineering Team

2025-12-25

Executive Summary

ElectricSQL Local-First Sync untuk app_barcode

1. Problem Statement

Current Pain Points

```
┌─────────────────────────────────────────────────────────────┐
│                    MASALAH SAAT
INI                           │
├─────────────────────────────────────────────────────────────┤
│
│
│  ❌ Offline = Tidak bisa scan
barcode                            │
│  ❌ Koneksi lambat = User
frustasi                               │
│  ❌ Data besar = Load time
```

```
lama                                │
│   ❌ Multi-device = Sync
conflict                               │
│   ❌ Server down = Operasi berhenti
total                         │
│
│
└──────────────────────────────────────────────────────────────┘
```

Business Impact

Issue	Impact
Downtime 1 jam	~Rp XXX juta lost productivity
Offline di gudang	Pairing barcode manual, error-prone
Slow sync	User bypass system, data inconsistent

---

2. Proposed Solution

ElectricSQL Local-First Architecture

```
┌──────────────────────────────────────────────────────────────┐
│                    SOLUSI: LOCAL-
FIRST                          │
├──────────────────────────────────────────────────────────────┤
│
│
│   ✅ Offline = Full functionality (local
SQLite)                │
│   ✅ Online = Auto-sync
background                               │
│   ✅ Conflict = CRDT auto-
resolve                               │
```

```
│  ✅ Large data = Shape-based partial
sync                       │
│  ✅ Server down = Client tetap
jalan                           │
│
│
└──────────────────────────────────────────────────────────────┘
```

---

3. Why ElectricSQL?

Comparison Matrix

Feature	ElectricSQL	PowerSync	Custom Sync
PostgreSQL Native	✅ Yes	✅ Yes	⚠️ Manual
CRDT Conflict Resolution	✅ Built-in	❌ Last-write-wins	❌ Manual
Open Source	✅ Apache 2.0	⚠️ Partial	N/A
Partial Sync (Shapes)	✅ Yes	✅ Yes	⚠️ Manual
Real-time Updates	✅ WebSocket	✅ WebSocket	⚠️ Manual
Active Development	✅ Supabase backing	✅ Active	N/A

Key Advantage: CRDT

CRDT (Conflict-free Replicated Data Types) memungkinkan: - Multiple users edit sama data offline - Auto-merge tanpa conflict - Eventual consistency guaranteed

```
User A (Offline): Scan barcode → Pair ke Parent X
User B (Offline): Scan barcode → Update note "QC passed"
                    ↓
             [Both go online]
                    ↓
        CRDT merges: Parent X + Note "QC passed"
              (No conflict, no data loss)
```

---

4. High-Level Architecture

```
┌─────────────────────────────────────────────────────────────────────┐
│                         ARCHITECTURE
OVERVIEW                        │
├─────────────────────────────────────────────────────────────────────┤
│
│
│   ┌──────────────┐
┌──────────────────┐                     │
│   │ PostgreSQL   │◄────────►│ Electric Service
│                    │
│   │ app_barcode  │  WAL     │ (Sync Engine)
│                    │
│   │              │  Capture │
│                    │
│   └──────────────┘
└────────┬─────────┘                     │
│
│                               │
│                                     │ WebSocket /
```

```
HTTP              │
│
│                               │
│
┌──────────────────────┼──────────────────────┐       │
│              │
│                      │        │
│              ▼
▼                      ▼        │
│   ┌─────────────────┐    ┌─────────────────┐
┌─────────────────┐│
│   │ Mobile App      │    │ Desktop App     │    │ Web
App         ││
│   │ (SQLite)        │    │ (SQLite)        │    │
(IndexedDB)     ││
│   │                 │    │                 │
│                 ││
│   │ Scanner A       │    │ Admin Console   │    │
Dashboard       ││
│   └─────────────────┘    └─────────────────┘
└─────────────────┘│
│
│
└────────────────────────────────────────────────────────────────────┘
```

5. Scope of Implementation

In Scope (Phase 1)

Table	Priority	Reason
`bc_batch`	HIGH	Master data, small
`bc_parent`	HIGH	Container hierarchy

Table	Priority	Reason
`bc_barcode`	CRITICAL	Core scanning data
`bc_token`	HIGH	Validation tokens
`bc_mfg`	MEDIUM	Manufacturing tracking
`bc_pair_brcdxparent`	MEDIUM	Pairing logs

Out of Scope (Phase 1)

Table	Reason
`bc_logs`	Audit only, server-side
`bc_downloads`	File management, server-side
Views (`view_*`)	Computed on client

---

6. Success Metrics

Technical KPIs

Metric	Target	Measurement
Offline Capability	100% core features	Manual testing
Initial Sync Time	< 30 seconds	P95 latency
Incremental Sync	< 2 seconds	P95 latency
Conflict Resolution	99.9% auto-resolved	Conflict logs
Data Consistency	100% eventual	Checksum validation

Business KPIs

Metric	Target	Current
Scan Success Rate	99.9%	TBD
User Productivity	+20%	Baseline TBD
Support Tickets (sync issues)	-50%	Baseline TBD

---

7. Risk Summary

Risk	Severity	Mitigation
Data loss during sync	HIGH	CRDT + versioning + backup
Security breach	HIGH	E2E encryption + RLS
Performance degradation	MEDIUM	Batching + indexing
User adoption	MEDIUM	Training + gradual rollout

Detail analisis di 07_BLINDSPOTS.md

---

8. Timeline Estimate

```
┌─────────────────────────────────────────────────────────────┐
│                    IMPLEMENTATION
TIMELINE                   │
├─────────────────────────────────────────────────────────────┤
│
│
│  Week 1-2: Setup &
```

```
PoC                                       │
│  ├── Electric service
deployment                             │
│  ├── Basic sync for
bc_batch                                 │
│  └── Client SDK
integration                                  │
│
│
│  Week 3-4: Core
Implementation                               │
│  ├── Full schema sync (bc_barcode, bc_parent,
bc_token)     │
│  ├── Conflict resolution
rules                               │
│  └── Offline
testing                                         │
│
│
│  Week 5-6: Security &
Optimization                           │
│  ├── Row-level
security                                      │
│  ├── Performance
tuning                                      │
│  └── Load
testing                                            │
│
│
│  Week 7-8:
Rollout                                           │
│  ├── Beta testing (1
vendor)                                 │
│  ├── Monitoring &
fixes                                      │
│  └── Gradual
rollout                                         │
```

```
│
│
└──────────────────────────────────────────────────────────────────────────┘
```

9. Recommendation

> **PROCEED** dengan ElectricSQL implementation dengan catatan:
>
> 1. ✅ Start dengan PoC untuk 1 vendor
> 2. ✅ Implement security layer first
> 3. ✅ Setup monitoring sebelum production
> 4. ⚠️ Plan rollback strategy
> 5. ⚠️ Train users on offline behavior

10. Next Steps

1. **Immediate**: Review Architecture dan Security
2. **Week 1**: Setup Electric service di staging
3. **Week 2**: PoC dengan subset data
4. **Week 3+**: Iterative implementation

System Architecture

ElectricSQL Local-First Sync

1. Architecture Overview

1.1 High-Level System Design

```
┌──────────────────────────────────────────────────────────────────────
│                           PRODUCTION
ENVIRONMENT                             │
├──────────────────────────────────────────────────────────────────────
│
│
│
┌─────────────────────────────────────────────────────────────────────┐
│
│  │                         DATA
LAYER                                   │    │
│  │  ┌─────────────┐     ┌─────────────┐
┌─────────────┐            │    │
│  │  │ PostgreSQL  │────▶│ Electric    │────▶│ Redis
│            │    │
│  │  │ Primary     │ WAL │ Sync Engine │     │ Cache
│            │    │
│  │  │             │     │             │     │
│            │    │
│  │  └──────┬──────┘     └──────┬──────┘
└─────────────┘            │    │
│  │         │
│                                        │    │
│  │         ▼
│                                        │    │
│  │  ┌─────────────┐
│                                        │    │
│  │  │ PostgreSQL  │
│                                        │    │
│  │  │ Replica     │
│                                        │    │
│  │  └─────────────┘
│                                        │    │
```

```
│
└────────────────────────────┼────────────────────────────────────────┘
│
│
│                                              │
│
┌────────────────────────────┼────────────────────────────────────────┐
│
│  │                         API
LAYER                                    │    │
│  │
│                                         │    │
│  │  ┌─────────────┐     ┌─────┴─────┐
┌─────────────┐              │    │
│  │  │ FastAPI     │     │ Electric  │     │ Auth
│              │    │
│  │  │ Backend     │     │ Proxy     │     │ Service
│              │    │
│  │  │ (REST)      │     │ (WS/HTTP) │     │ (JWT)
│              │    │
│  │  └─────────────┘     └─────┬─────┘
└─────────────┘              │    │
│
└────────────────────────────┼────────────────────────────────────────┘
│
│
│                                              │
│
┌────────────────────────────┼────────────────────────────────────────┐
│
│  │                      DELIVERY
LAYER                                  │    │
│  │
│                                         │    │
│  │  ┌─────────────┐     ┌─────┴─────┐
┌─────────────┐              │    │
│  │  │ CDN         │     │ Load      │     │ API
│              │    │
│  │  │ (Static)    │     │ Balancer  │     │ Gateway
│              │    │
│  │  └─────────────┘     └───────────┘
└─────────────┘              │    │
│
└─────────────────────────────────────────────────────────────────────┘
│
│
│
└──────────────────────────────────────────────────────────────────────
                                    │
                                    │ Internet
                                    │
┌──────────────────────────────────────────────────────────────────────
│                              CLIENT
LAYER                                    │
├──────────────────────────────────────────────────────────────────────
│
│
│  ┌─────────────────┐   ┌─────────────────┐
┌─────────────────┐           │
│  │   MOBILE APP    │   │   DESKTOP APP   │   │    WEB
APP      │           │
│  │  (React Native) │   │   (Electron)    │   │   (Vue/
React)   │           │
│  │                 │   │                 │
│                 │           │
│  │  ┌───────────┐  │   │  ┌───────────┐  │   │
┌───────────┐  │           │
│  │  │ Electric  │  │   │  │ Electric  │  │   │  │
Electric  │  │           │
│  │  │ Client    │  │   │  │ Client    │  │   │  │
Client    │  │           │
│  │  └─────┬─────┘  │   │  └─────┬─────┘  │   │
└─────┬─────┘  │           │
│  │        │        │   │        │        │   │
│        │           │
│  │  ┌─────┴─────┐  │   │  ┌─────┴─────┐  │   │
┌─────┴─────┐  │           │
│  │  │ SQLite    │  │   │  │ SQLite    │  │   │  │
IndexedDB │  │           │
│  │  │ (Local)   │  │   │  │ (Local)   │  │   │  │
(Local)   │  │           │
│  │  └───────────┘  │   │  └───────────┘  │   │
└───────────┘  │           │
│  └─────────────────┘   └─────────────────┘
└─────────────────┘           │
│
│
└──────────────────────────────────────────────────────────────
```

2. Component Details

2.1 Electric Sync Engine

```
┌─────────────────────────────────────────────────────────────┐
│                    ELECTRIC SYNC
ENGINE                      │
├─────────────────────────────────────────────────────────────┤
│
│
│
┌─────────────────────────────────────────────────────┐
│
│  │                  WAL CONSUMER
│    │
│  │  • Reads PostgreSQL Write-Ahead Log
│    │
│  │  • Captures INSERT, UPDATE, DELETE
│    │
│  │  • Maintains replication slot
│    │
│
└────────────────────────────────────────────────────────┘
                        ┬
│
│
│                                      │
│
▼                                      │
│
┌────────────────────────────────────────────────────────┐
│
│  │                  SHAPE MANAGER
│    │
│  │  • Defines data subsets for sync
│    │
│  │  • Filters by vendor_code, batch_id
│    │
│  │  • Manages shape subscriptions
│    │
│
└────────────────────────────────────────────────────────┘
                        ┬
│
│
│                                      │
│
▼                                      │
│
┌────────────────────────────────────────────────────────┐
│
│  │                  CRDT PROCESSOR
│    │
│  │  • Converts SQL ops to CRDT operations
│    │
│  │  • Handles conflict detection
│    │
│  │  • Generates merge operations
│    │
```

```
│
└──────────────────────┬──────────────────────────────┘
│
│
│                                    │
│
▼                                    │
│
┌─────────────────────────────────────────────────────┐
│
│  │                   SYNC DISPATCHER
│    │
│  │  • WebSocket connections to clients
│    │
│  │  • HTTP long-polling fallback
│    │
│  │  • Batches and compresses payloads
│    │
│
└─────────────────────────────────────────────────────┘
│
│
│
└─────────────────────────────────────────────────────────────┘
```

2.2 Client Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    CLIENT
APPLICATION                        │
├─────────────────────────────────────────────────────────────┤
│
│
│
┌─────────────────────────────────────────────────────┐
│
│  │                   UI LAYER
│    │
│  │  • Barcode Scanner Component
│    │
│  │  • Batch Management UI
│    │
│  │  • Sync Status Indicator
│    │
│
└───────────────────────┬──────────────────────────────┘
│
│
│                                     │
│
▼                                     │
│
┌──────────────────────────────────────────────────────┐
│
│  │                 SYNC LAYER
│    │
│  │  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐
│    │
│  │  │ Electric     │  │ Offline      │  │ Conflict     │
│    │
│  │  │ Client SDK   │  │ Queue        │  │ Resolver     │
│    │
│  │  └──────────────┘  └──────────────┘  └──────────────┘
│    │
│
└───────────────────────┬──────────────────────────────┘
│
│
│                                     │
│
▼                                     │
│
┌──────────────────────────────────────────────────────┐
│
```

```
│  │                 DATA LAYER
│    │
│  │  ┌───────────────────────────────────────────────┐
│    │
│  │  │              LOCAL DATABASE                     │
│    │
│  │  │  • SQLite (Mobile/Desktop)                      │
│    │
│  │  │  • IndexedDB (Web)                              │
│    │
│  │  │  • Encrypted at rest                            │
│    │
│  │  └───────────────────────────────────────────────┘
│    │
│
└───────────────────────────────────────────────────────┘
│
│
│
└────────────────────────────────────────────────────────────────┘
```

3. Data Flow Diagrams

3.1 Initial Sync Flow

```
┌──────────────────────────────────────────────────────────────────────
│                        INITIAL SYNC
FLOW                                  │
├──────────────────────────────────────────────────────────────────────
│
│
│  CLIENT                    ELECTRIC
POSTGRESQL        │
│    │
│                            │              │
│    │  1. Connect + Auth
│                             │             │
│
│─────────────────────────▶│
│             │
│    │
│                             │             │
│    │  2. Subscribe Shape
│                             │             │
│    │  (vendor_code=X)
│                             │             │
│
│─────────────────────────▶│
│             │
│    │
│                             │             │
│    │                          │  3. Query Initial
Data     │             │
│    │
│───────────────────────────▶│             │
│    │
│                             │             │
│    │                          │  4. Return Rows
(batched)  │             │
│    │
│◀───────────────────────────│             │
│    │
│                             │             │
│    │  5. Stream Data Chunks
│                             │             │
│
│◀─────────────────────────│
│             │
│    │  (compressed, CRDT ops)
│                             │             │
│    │
│                             │             │
│    │  6. Insert to Local DB
│                            │             │
│    ├─────────┐
│                            │             │
│    │         │
│                            │             │
│    │◀────────┘
│                            │             │
│    │
│                            │             │
│    │  7. Sync Complete
│                            │             │
│
│◀─────────────────────────│
│             │
│    │
│                            │             │
│
│
└───────────────────────────────────────────────────────────────────────
```

3.2 Real-time Sync Flow

```
┌───────────────────────────────────────────────────────────────────────
│                       REAL-TIME SYNC
FLOW                                │
├───────────────────────────────────────────────────────────────────────
│
│
│  POSTGRESQL                   ELECTRIC
CLIENT           │
│    │
│                            │             │
│    │  1. INSERT/UPDATE row
│                            │             │
│
├─────────────────────────▶│
```

```
│             │
│    │  (via FastAPI backend)   │ (WAL
capture)              │             │
│    │
│                            │             │
│    │                          │  2. Check Shape
Match      │             │
│    │
├─────────┐                  │             │
│    │                          │
│                  │             │
│    │
│◀────────┘                  │             │
│    │
│                            │             │
│    │                          │  3. Push to
Subscribers    │             │
│    │
├───────────────────────────▶│             │
│    │                          │  (WebSocket, CRDT
op)      │             │
│    │
│                            │             │
│    │
│                            │  4. Apply   │
│    │
│                            │  to Local   │
│    │
│                            ├─────────┐   │
│    │
│                            │         │   │
│    │
│                            │◀────────┘   │
│    │
│                            │             │
│    │                          │  5.
ACK                    │             │
```

```
│    │
│◄────────────────────────────│             │
│    │
│                            │             │
│
│
└──────────────────────────────────────────────────────────────────────
```

3.3 Offline → Online Sync Flow

```
┌──────────────────────────────────────────────────────────────────────
│                    OFFLINE → ONLINE SYNC
FLOW                            │
├──────────────────────────────────────────────────────────────────────
│
│
│  ═══════════════════════ OFFLINE PHASE
═══════════════════════          │
│
│
│  CLIENT
(Offline)
│
│
│
│
│    │  1. User scans
barcode                                                    │
│
├─────────┐
│
│    │
│
│
│    │  2. Write to local
SQLite                                             │
│    │         │  + Add to pending_ops
```

```
queue                               │
│
│◄────────┘
│
│
│
│
│    │  3. User continues working
offline...                               │
│    │  (All ops queued
locally)                                           │
│
│
│
│
│
│  ═══════════════════════ COMES ONLINE
═══════════════════════           │
│
│
│  CLIENT                    ELECTRIC
POSTGRESQL        │
│    │
│                            │             │
│    │  4. Reconnect
│                            │             │
│
│─────────────────────────►│
│             │
│    │
│                            │             │
│    │  5. Pull server changes
│                            │             │
│
│◄─────────────────────────│
│             │
│    │  (CRDT ops since last)
│                            │             │
│    │
│                            │             │
│    │  6. CRDT Merge
│                            │             │
│    │  (local + remote)
│                            │             │
│    │─────────┐
│                            │             │
│    │         │
│                            │             │
│    │◄────────┘
│                            │             │
│    │
│                            │             │
│    │  7. Push pending ops
│                            │             │
│
│─────────────────────────►│
│             │
│    │
│                            │             │
│    │                          │  8. Write to
PostgreSQL    │             │
│    │
│───────────────────────────►│             │
│    │
│                            │             │
│    │  9. Confirm sync
│                            │             │
│
│◄─────────────────────────│
│             │
│    │
│                            │             │
│
```

```
│
└──────────────────────────────────────────────────────────────────────
```

4. Shape Definitions

4.1 Shape Strategy per Multi-Tenant

```
┌──────────────────────────────────────────────────────────────────────┐
│                     SHAPE
DEFINITIONS                                    │
├──────────────────────────────────────────────────────────────────────┤
│
│
│  SHAPE 1: Vendor Master
Data                                            │
│
─────────────────────────────
│
│  Tables: bc_batch, bc_mfg,
bc_config                                    │
│  Filter: vendor_code
= :current_vendor                                  │
│  Sync: Always,
Full                                                    │
│
│
│
┌──────────────────────────────────────────────────────┐
│
│  │  Shape ID: vendor_master_{vendor_code}
│    │
│  │  WHERE: vendor_code = 'VENDOR001'
│    │
│  │         AND is_deleted = false
│    │
```

```
│
└──────────────────────────────────────────────────────┘
│
│
│
│  SHAPE 2: Active Batch
Data                                  │
│
──────────────────────────
│
│  Tables: bc_parent, bc_barcode,
bc_token                     │
│  Filter: vendor_code + batch_id (active
only)                │
│  Sync: On-demand per batch
assignment                        │
│
│
│
┌──────────────────────────────────────────────────────┐
│
│  │  Shape ID: batch_data_{vendor_code}_{batch_id}
│    │
│  │  WHERE: vendor_code = 'VENDOR001'
│    │
│  │         AND batch_id = 123
│    │
│  │         AND is_deleted = false
│    │
│
└──────────────────────────────────────────────────────┘
│
│
│
│  SHAPE 3: Recent Pairing
Logs                                │
│
```

```
────────────────────────────
│
│  Tables: bc_pair_brcdxparent,
bc_inbound_pairing             │
│  Filter: vendor_code + created_at (last 7
days)                │
│  Sync: Background,
incremental                         │
│
│
│
┌──────────────────────────────────────────────────────┐
│
│  │  Shape ID: pairing_logs_{vendor_code}
│    │
│  │  WHERE: vendor_code = 'VENDOR001'
│    │
│  │         AND created_at > now() - interval '7 days'
│    │
│
└──────────────────────────────────────────────────────┘
│
│
│
└──────────────────────────────────────────────────────────────┘
```

4.2 Shape Subscription Lifecycle

```
┌──────────────────────────────────────────────────────────────┐
│                SHAPE SUBSCRIPTION
LIFECYCLE                    │
├──────────────────────────────────────────────────────────────┤
│
│
│  ┌──────────┐     ┌──────────┐
┌──────────┐              │
│  │  LOGIN   │────►│ SUBSCRIBE│────►│  ACTIVE
│             │
│  │          │     │  SHAPES  │     │
│             │
│  └──────────┘     └──────────┘
└─────┬────┘             │
│
│                   │
│
▼                   │
│
┌───────────────────┐          │
│                              │
│          │
│                              ▼
│          │
│                   ┌──────────────────┐
│           │
│                   │  User switches   │
│           │
│                   │  to new batch    │
│           │
│                   └────────┬─────────┘
│           │
│                            │
│           │
│                            ▼
│           │
│                   ┌──────────────────┐
│           │
│                   │  Subscribe new   │
│           │
│                   │  batch shape     │
│           │
│                   └────────┬─────────┘
│           │
│                            │
│           │
```

```
│
└──────────────────────┘          │
│
│
│  Note: Old shapes stay subscribed for quick
switching        │
│        LRU eviction for memory
management                    │
│
│
└──────────────────────────────────────────────────────────────┘
```

5. Deployment Architecture

5.1 Infrastructure Stack

```
┌──────────────────────────────────────────────────────────────
│                       DEPLOYMENT
ARCHITECTURE                                │
├──────────────────────────────────────────────────────────────
│
│
│
┌──────────────────────────────────────────────────────────────┐
│
│  │                        KUBERNETES
CLUSTER                     │    │
│
│
│    │
│  │  ┌──────────────────┐  ┌──────────────────┐
┌──────────────────┐   │    │
│  │  │ Electric Pod     │  │ Electric Pod     │  │ Electric
Pod     │  │    │
│  │  │ (Replica 1)      │  │ (Replica 2)      │  │ (Replica
3)      │  │    │
│  │  └─────────────────┬─────────────────┘  └─────────────────┬─────────────────┘
└─────────────────┬─────────────────┘  │    │
│  │           │                    │
│           │    │
│  │
└────────────────────┼────────────────────┘           │
│
│  │
│                                │    │
│  │
┌──────┴──────┐                          │    │
│  │                           │   Service
│                         │    │
│  │                           │ (ClusterIP)
│                         │    │
│  │
└──────┬──────┘                          │    │
│  │
│                                │    │
│
└────────────────────────────────┼────────────────────────────────┘
│
│
│                                      │
│
┌────────────────────────────────┴────────────────────────────────┐
│
│  │                           LOAD
BALANCER                               │    │
│  │
┌─────────────────────────────────────────────────────────┐
│    │
│  │  │  • SSL
Termination                                          │    │
│
│  │  │  • WebSocket Sticky
```

```
Sessions                             │    │    │
│  │  │  • Health
Checks                                         │    │    │
│  │
└─────────────────────────────────────────────────────────┘
│    │
│
└─────────────────────────────────────────────────────────────────┘
│
│
│
│
┌─────────────────────────────────────────────────────────────────┐
│
│  │                        DATABASE
LAYER                            │    │
│
│
│    │
│  │  ┌─────────────────┐
┌─────────────────┐           │    │
│  │  │ PostgreSQL      │──────────────│ PostgreSQL
│           │    │
│  │  │ Primary         │  Streaming   │ Replica
│           │    │
│  │  │ (RDS/CloudSQL)  │  Replication │ (Read-only)
│           │    │
│  │  └─────────────────┘
└─────────────────┘           │    │
│
│
│    │
│
└─────────────────────────────────────────────────────────────────┘
│
│
```

```
│
└──────────────────────────────────────────────────────────
```

5.2 Docker Compose (Development)

```yaml
# docker-compose.electric.yaml

version: '3.8'

services:
  postgres:
    image: postgres:16
    environment:
      POSTGRES_DB: barcode
      POSTGRES_USER: electric
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    command:
      - "postgres"
      - "-c"
      - "wal_level=logical"  # Required for Electric
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  electric:
    image: electricsql/electric:latest
    environment:
      DATABASE_URL: postgresql://electric:${DB_PASSWORD}@postgres:5432/barcode
      ELECTRIC_WRITE_TO_PG_MODE: direct_writes
      AUTH_MODE: secure
      AUTH_JWT_KEY: ${JWT_SECRET}
    ports:
      - "5133:5133"  # Electric HTTP API
      - "5433:5433"  # Electric Proxy (Postgres wire protocol)
    depends_on:
      - postgres

  fastapi:
    build: ./backend
    environment:
      DATABASE_URL: postgresql://app:${DB_PASSWORD}
        @postgres:5432/barcode
      ELECTRIC_URL: http://electric:5133
    ports:
      - "8000:8000"
    depends_on:
      - electric

volumes:
  postgres_data:
```

---

6. Network Architecture

6.1 Network Topology

```
┌──────────────────────────────────────────────────────────────
│                          NETWORK
TOPOLOGY                                     │
├──────────────────────────────────────────────────────────────
│
│
│
INTERNET                                           │
│
│                                               │
│
▼                                               │
│
┌──────────────────────────────────────────────────────────────
```

```
│
│  │                        CDN /
WAF                                 │      │
│  │  • CloudFlare / AWS
CloudFront                                 │      │
│  │  • DDoS
protection
│      │
│  │  • SSL/TLS
termination                                         │
│
│
└───────────────────────────────────────────────────────────────┘
│
│
│                                           │
│
▼                                           │
│
┌───────────────────────────────────────────────────────────────┐
│
│  │                      VPC / Private
Network                        │      │
│
│
│      │
│  │  Public
Subnet
│      │
│  │
┌─────────────────────────────────────────────────────┐
│      │
│  │  │  Load Balancer (ALB/NLB)
│      │      │
│  │  │  • /api/* → FastAPI
│      │      │
│  │  │  • /v1/shape/* → Electric
│      │      │
│  │  │  • WebSocket → Electric (sticky session)
│      │      │
│  │
└────────────────────────────────────────────────────┘
│      │
│  │
│                                 │      │
│  │  Private Subnet
│                                 │      │
│  │
┌───────────────────────────┼───────────────────────┐
│      │
│  │  │                           │
│        │      │
│  │  │  ┌─────────┐       ┌──────┴──────┐
┌───────┐│        │      │
│  │  │  │ FastAPI │       │  Electric   │      │ Redis
││        │      │
│  │  │  │ Pods    │       │  Pods       │      │
││        │      │
│  │  │  └─────────┘       └─────────────┘
└───────┘│        │      │
│  │  │
│        │      │
│  │
└───────────────────────────────────────────────────┘
│      │
│  │
│                                 │      │
│  │  Database Subnet
│                                 │      │
│  │
┌───────────────────────────┼───────────────────────┐
│      │
│  │  │                           │
│        │      │
│   │   │   ┌──────────────┐     ┌─────┴──────┐
│           │      │
│   │   │   │ PostgreSQL   │     │ PostgreSQL│
│           │      │
│   │   │   │ Primary      │────►│ Replica    │
│           │      │
│   │   │   └──────────────┘     └────────────┘
│           │      │
│   │   │
│           │      │
│   │
└─────────────────────────────────────────────────────┘
│      │
│
│
│      │
│
└──────────────────────────────────────────────────────────────────┘
│
│
│
└──────────────────────────────────────────────────────────────────────
```

7. Monitoring Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    OBSERVABILITY
STACK                            │
├─────────────────────────────────────────────────────────────┤
│
│
│
┌───────────────────────────────────────────────────────┐
│
│   │                         COLLECTION
│    │
│  │  ┌─────────┐  ┌─────────┐  ┌─────────┐
│    │
│  │  │ Metrics │  │ Logs    │  │ Traces  │
│    │
│  │  │ (Prom.) │  │ (Loki)  │  │ (Jaeger)│
│    │
│  │  └────┬────┘  └────┬────┘  └────┬────┘
│    │
│
└───────┼────────────┼────────────┼────────────────────┘
│
│          │            │
│                         │
│          ▼            ▼
▼                         │
│
┌─────────────────────────────────────────────────────┐
│
│  │                   STORAGE
│    │
│  │  ┌─────────────────────────────────────────────┐
│    │
│  │  │             Grafana Stack                    │
│    │
│  │  │  • Mimir (Metrics)                          │
│    │
│  │  │  • Loki (Logs)                              │
│    │
│  │  │  • Tempo (Traces)                           │
│    │
│  │  └─────────────────────────────────────────────┘
│    │
│
└─────────────────────────────────────────────────────┘
│
│
```

```
│                                    │
│
▼                                    │
│
┌─────────────────────────────────────────────────────┐
│
│  │                  VISUALIZATION
│    │
│  │  ┌─────────────────────────────────────────────┐
│    │
│  │  │             Grafana Dashboards               │
│    │
│  │  │  • Sync Performance                          │
│    │
│  │  │  • Error Rates                               │
│    │
│  │  │  • Client Connections                        │
│    │
│  │  │  • Database Health                           │
│    │
│  │  └─────────────────────────────────────────────┘
│    │
│
└─────────────────────────────────────────────────────┘
│
│
│                                    │
│
▼                                    │
│
┌─────────────────────────────────────────────────────┐
│
│  │                    ALERTING
│    │
│  │  • PagerDuty / OpsGenie
│    │
│  │  • Slack / Telegram notifications
│       │
│
└──────────────────────────────────────────────────────────────────────────┘
│
│
│
└─────────────────────────────────────────────────────────────────────────────────────┘
```

Sync Algorithm

ElectricSQL CRDT-Based Synchronization

---

1. Core Principles

```
┌─────────────────────────────────────────────────────────────────────────────────────┐
│                          SYNC ALGORITHM
PRINCIPLES                          │
├─────────────────────────────────────────────────────────────────────────────────────┤
│
│
│   1. LOCAL-FIRST: All reads/writes go to local DB
first       │
│   2. EVENTUAL CONSISTENCY: All replicas
converge                   │
│   3. CONFLICT-FREE: CRDT auto-merge, no manual
resolution     │
│   4. PARTIAL REPLICATION: Only sync what user
needs            │
│
```

```
│
└──────────────────────────────────────────────────────────────────────┘
```

2. CRDT Types Used

Field Type	CRDT	Resolution
`note`, `status`	LWW-Register	Latest timestamp wins
`is_deleted`, `is_archived`	Flag (OR)	True always wins
Pairing logs	OR-Set	All pairings kept, deduped
Counters	G-Counter	Sum all increments

CRDT Merge Example

```
Device A (Offline): barcode.note = "QC OK" @ T1
Device B (Offline): barcode.parent_id = 500 @ T2

[Both come online → CRDT Merge]

Result: { note: "QC OK", parent_id: 500 }
✅ Both changes preserved, no conflict!
```

3. Sync Phases

```
PHASE 1: BOOTSTRAP (First-time)
├── Authenticate → Get JWT
├── Subscribe vendor_master shape
├── Download bc_batch, bc_config
├── For each assigned batch:
```

```
│    └── Subscribe batch_data shape
└── Mark bootstrap complete

PHASE 2: REAL-TIME (Continuous)
├── WebSocket receives CRDT op from Electric
├── Apply to local SQLite
├── Update UI reactively
└── Send ACK

PHASE 3: PUSH (Local → Server)
├── Write to local SQLite (immediate)
├── Queue CRDT operation
├── If online: Push immediately
└── If offline: Queue for later

PHASE 4: CATCHUP (After offline)
├── Get server checkpoint
├── Fetch ops since local checkpoint
├── CRDT merge with local
└── Push pending ops
```

4. Batching Strategy

Tier	Data	Rows	Sync Timing
CRITICAL	bc_config, bc_batch (active)	~1K	Always, on app start
WORKING SET	bc_parent, bc_barcode per batch	~21K/batch	On-demand
HISTORICAL	Completed batches	Variable	Background, low priority

Tier	Data	Rows	Sync Timing
NEVER SYNC	bc_logs, views	N/A	Server-only

Chunked Download

```
CHUNK_SIZE = 10000 rows
MAX_CONCURRENT = 3 chunks

FOR each chunk:
  1. Fetch chunk from Electric
  2. Write to SQLite in transaction
  3. Emit progress update
```

---

5. Offline Queue

```
CREATE TABLE pending_operations (
  id INTEGER PRIMARY KEY,
  operation_type TEXT,
  table_name TEXT,
  row_id INTEGER,
  crdt_operation BLOB,
  created_at TIMESTAMP,
  retry_count INTEGER DEFAULT 0,
  status TEXT DEFAULT 'pending'
);
```

Queue Processing

```
FUNCTION process_pending_queue():
  pending = SELECT * FROM pending_operations WHERE
status='pending'
```

```
  FOR op IN pending:
    TRY:
      electric.push(op.crdt_operation)
      DELETE op
    CATCH NetworkError:
      BREAK  // Stop, still offline
    CATCH Error:
      IF retry_count >= 5:
        status = 'failed'
      ELSE:
        retry_count += 1
```

---

6. Conflict Resolution Rules

Table	Field	Rule
bc_barcode	parent_id	First-write-wins (one barcode = one parent)
bc_barcode	note	Last-write-wins
bc_barcode	is_deleted	True always wins
bc_pair_*	(row)	OR-Set, keep all unique pairings

Business Rule: Barcode Pairing Conflict

```
IF concurrent pairing to different parents:
  1. Keep FIRST pairing (earlier timestamp)
  2. Reject second pairing
  3. Notify user of rejection
```

---

7. Integrity Check

```
FUNCTION integrity_check(vendor_code):
  local = COUNT(*), SUM(id), MAX(updated_at) FROM bc_barcode
  server = electric.get_checksum(bc_barcode, vendor_code)

  IF local != server:
    force_resync(vendor_code)

Schedule: Every 6 hours when on WiFi
```

8. Complexity Analysis

Operation	Time	Space
Initial sync	O(N/C)	O(C) per chunk
Incremental sync	O(Δ)	O(Δ)
Single write	O(1)	O(1)
CRDT merge	O(1)	O(1)

Key insight: Incremental sync is proportional to CHANGES, not total data.

API Routing Design

ElectricSQL + FastAPI Integration

1. API Architecture Overview

```
┌─────────────────────────────────────────────────────────────┐
│                     API LAYER
ARCHITECTURE                    │
├─────────────────────────────────────────────────────────────┤
│
│
│
CLIENT
│
│
│                                                          │
│    ├──────▶ /api/* ─────▶ FastAPI Backend ─────▶
PostgreSQL   │
│    │       (Business Logic,
Validation)                    │
│
│                                                          │
│    └──────▶ /v1/shape/* ─────▶ Electric ─────▶
PostgreSQL     │
│            (Sync Operations via WebSocket/
HTTP)            │
│
│
└─────────────────────────────────────────────────────────────┘
```

---

2. Endpoint Categories

2.1 Authentication Endpoints

Method	Path	Description
POST	`/api/auth/login`	Login, returns JWT

Method	Path	Description
POST	`/api/auth/refresh`	Refresh JWT token
POST	`/api/auth/logout`	Invalidate token
GET	`/api/auth/me`	Get current user info

2.2 Electric Sync Endpoints

Method	Path	Description
GET	`/v1/shape`	Get shape definition
GET	`/v1/shape/{shape_id}`	Subscribe to shape (SSE/WS)
POST	`/v1/shape/{shape_id}/sync`	Initial sync for shape
WS	`/v1/shape/{shape_id}/live`	Real-time sync WebSocket

2.3 Business Logic Endpoints

Method	Path	Description
POST	`/api/barcode/scan`	Validate & process barcode scan
POST	`/api/barcode/pair`	Pair barcode to parent
GET	`/api/batch/{id}`	Get batch details
POST	`/api/batch/{id}/activate`	Activate batch

2.4 Sync Status Endpoints

Method	Path	Description
GET	`/api/sync/status`	Get sync status
POST	`/api/sync/force`	Force full resync
GET	`/api/sync/pending`	Get pending operations count

---

3. Shape Definitions

3.1 Shape: vendor_master

```yaml
# Subscription URL: /v1/shape/vendor_master?vendor_code=ABC
shape:
  name: vendor_master
  tables:
    - bc_batch
    - bc_config
    - bc_mfg
  params:
    - vendor_code
  where: |
    vendor_code = :vendor_code
    AND is_deleted = false
  permissions:
    read: vendor_member
    write: vendor_admin
```

3.2 Shape: batch_data

```yaml
# Subscription URL: /v1/shape/batch_data?batch_id=123
shape:
  name: batch_data
  tables:
    - bc_parent
    - bc_barcode
    - bc_token
  params:
    - batch_id
  where: |
    batch_id = :batch_id
    AND is_deleted = false
  permissions:
    read: batch_assigned
    write: batch_operator
```

3.3 Shape: pairing_logs

```yaml
# Subscription URL: /v1/shape/pairing_logs?vendor_code=ABC
shape:
  name: pairing_logs
  tables:
    - bc_pair_brcdxparent
    - bc_inbound_pairing
  params:
    - vendor_code
  where: |
    vendor_code = :vendor_code
    AND created_at > now() - interval '7 days'
  permissions:
    read: vendor_member
    write: batch_operator
```

---

4. Request/Response Flow

4.1 Initial Sync Flow

```
Client                    API Gateway
Electric              PostgreSQL
  │                            │
│                       │
  │ 1. POST /api/auth/login    │
│                       │
  ├───────────────────────────►│
│                       │
  │                            │ validate credentials
│                       │
  │
├───────────────────────────────────────────────►│
  │                            │
│                       │
  │ ◄──────────────────────────│ JWT token
│                       │
  │                            │
│                       │
  │ 2. GET /v1/shape/vendor_master?vendor_code=ABC
│                       │

├─────────────────────────►├──────────────────────►│
│
  │                            │                        │
query shape data      │
  │                            │
├─────────────────────►│
  │                            │
│                       │
  │ ◄──────────────────────────────────────────────────│
stream data chunks    │
  │ (chunked response)         │
│                       │
```

```
   │                           │
 │                       │
   │ 3. WS /v1/shape/vendor_master/live
 │                       │

 │──────────────────────────▶│────────────────────────▶│
 │
   │◀══════════════════════════════════════════════════│
WebSocket connected   │
   │ (real-time updates)       │
 │                       │
```

4.2 Barcode Scan Flow (Online)

```
Client                         FastAPI
Electric                   PostgreSQL
   │                           │
 │                       │
   │ 1. Scan barcode locally   │
 │                       │
   │ (write to SQLite)         │
 │                       │
   │                           │
 │                       │
   │ 2. POST /api/barcode/scan │
 │                       │
   │ {barcode: "ABC123"}       │
 │                       │
   │──────────────────────────▶│
 │                       │
   │                           │ validate barcode
 │                       │
   │
 │─────────────────────────────────────────────────▶│
   │                           │
 │                       │
   │                           │ log to bc_logs
 │                       │
   │
 │──────────────────────────────────────────────►│
   │                           │
 │                       │
   │                           │                        │ WAL
 trigger                 │
   │                           │
 │◄──────────────────────│
   │                           │
 │                       │
   │ ◄─────────────────────────│ validation result
 │                       │
   │                           │
 │                       │
   │ ◄══════════════════════════════════════════════════│
 sync update to other │
   │ (WebSocket update)        │                        │
 devices              │
```

4.3 Offline → Online Sync Flow

```
Client (comes online)     FastAPI
Electric              PostgreSQL
  │                           │
│                       │
  │ 1. GET /v1/shape/batch_data?since=<checkpoint>
│                       │

│──────────────────────────►│───────────────────────►│
│
  │                           │                        │
fetch deltas         │
  │                           │
│──────────────────────►│
  │                           │
│                       │
```

```
  │ ◄──────────────────────────────────────────────────│
return changes       │
  │                            │
│                       │
  │ (CRDT merge locally)       │
│                       │
  │                            │
│                       │
  │ 2. POST /v1/shape/batch_data/write
│                       │
  │ {operations: [...]}        │
│                       │

│──────────────────────────►│───────────────────────►│
│
  │                            │                        │
apply CRDT ops       │
  │                            │
│──────────────────────►│
  │                            │
│                       │
  │ ◄──────────────────────────────────────────────────│
confirm              │
```

5. Authentication & Authorization

5.1 JWT Token Structure

```
{
  "sub": "user_123",
  "vendor_code": "ABC",
  "roles": ["operator", "viewer"],
  "assigned_batches": [1, 2, 3],
  "exp": 1735200000,
  "iat": 1735113600
}
```

5.2 Permission Model

Role	Shape Access	Write Access
`viewer`	vendor_master (read)	None
`operator`	batch_data (read/write)	bc_barcode, bc_pair_*
`admin`	All shapes	All tables

5.3 Row-Level Security

```
-- Electric applies these filters automatically
-- Users can only see data for their vendor_code

POLICY vendor_isolation ON bc_barcode
  USING (vendor_code = current_setting('app.vendor_code'));

POLICY batch_assignment ON bc_barcode
  USING (batch_id IN (
    SELECT batch_id FROM user_batch_assignments
    WHERE user_id = current_setting('app.user_id')
  ));
```

---

6. Error Handling

6.1 Error Response Format

```
{
  "error": {
    "code": "SYNC_CONFLICT",
    "message": "Barcode already paired to different
        parent",
    "details": {
      "barcode_id": 12345,
      "current_parent": 500,
      "attempted_parent": 600
    },
    "retry_after": null
  }
}
```

6.2 Error Codes

Code	HTTP	Description
`AUTH_EXPIRED`	401	Token expired, refresh needed
`AUTH_INVALID`	401	Invalid token
`PERMISSION_DENIED`	403	No access to resource
`SYNC_CONFLICT`	409	CRDT conflict (manual resolution needed)
`SHAPE_NOT_FOUND`	404	Shape definition not found
`RATE_LIMITED`	429	Too many requests
`SYNC_OUTDATED`	410	Shape version outdated, resync needed

---

7. Rate Limiting

Endpoint Type	Limit	Window
Auth endpoints	10 req	1 min
Shape sync	100 req	1 min

Endpoint Type	Limit	Window
Business logic	1000 req	1 min
WebSocket	1 connection	per device

---

8. WebSocket Protocol

8.1 Connection

```
// Client connects with JWT in query param
ws = new WebSocket('/v1/shape/batch_data/live?token=<jwt>&batch_id=123')
```

8.2 Message Types

Server → Client:

```
// Data update
{"type": "data", "table": "bc_barcode", "op": "UPDATE", "row": {...}}

// Checkpoint
{"type": "checkpoint", "lsn": "0/ABC123"}

// Heartbeat
{"type": "heartbeat", "ts": 1735200000}
```

Client → Server:

```
// ACK checkpoint
{"type": "ack", "lsn": "0/ABC123"}
```

```
// Push operation
{"type": "write", "table": "bc_barcode", "op": "UPDATE",
        "row": {...}}
```

Security Analysis

ElectricSQL Local-First Security

---

1. Security Overview

```
┌──────────────────────────────────────────────────────────────┐
│                     SECURITY
LAYERS                                 │
├──────────────────────────────────────────────────────────────┤
│
│
│  Layer 1: Network
Security                                            │
│  ├── TLS 1.3 for all
connections                                     │
│  ├── Certificate pinning
(mobile)                                   │
│  └── WAF / DDoS
protection                                          │
│
│
│  Layer 2:
Authentication                                             │
│  ├── JWT tokens with short
expiry                                   │
│  ├── Refresh token
rotation                                        │
│  └── Device
```

```
binding                                         │
│
│
│  Layer 3:
Authorization                                      │
│  ├── Role-based access control
(RBAC)                       │
│  ├── Row-level security (RLS) in
PostgreSQL                 │
│  └── Shape-level permissions in
Electric                    │
│
│
│  Layer 4: Data
Protection                                    │
│  ├── Encryption at rest (SQLite,
PostgreSQL)                │
│  ├── Encryption in transit
(TLS)                            │
│  └── Client-side encryption
(optional)                      │
│
│
│  Layer 5: Audit &
Monitoring                                 │
│  ├── All operations
logged                                  │
│  ├── Anomaly
detection                                      │
│  └── Compliance
reporting                                   │
│
│
└─────────────────────────────────────────────────────────────┘
```

2. Threat Model

2.1 STRIDE Analysis

Threat	Risk	Mitigation
Spoofing	Attacker impersonates user	JWT + device binding
Tampering	Data modification in transit	TLS + CRDT checksums
Repudiation	Deny sync operations	Audit logs + signed ops
Information Disclosure	Data leak	Encryption + RLS
Denial of Service	Overload sync service	Rate limiting + WAF
Elevation of Privilege	Access other vendor data	RLS + tenant isolation

2.2 Attack Vectors

```
┌───────────────────────────────────────────────────────────────┐
│                     ATTACK
VECTORS                              │
├───────────────────────────────────────────────────────────────┤
│
│
│  1. STOLEN
DEVICE                                             │
│
```

```
───────────────
│
│  Risk: Physical access to local SQLite
DB                   │
│  Impact: Data exposure for synced
barcodes                  │
│
Mitigation:
│
│  • SQLite encryption
(SQLCipher)                            │
│  • Remote wipe
capability                                   │
│  • Auto-logout after
inactivity                             │
│  • Biometric/PIN
lock                                       │
│
│
│  2. MAN-IN-THE-
MIDDLE                                        │
│
────────────────────
│
│  Risk: Intercept sync
traffic                               │
│  Impact: Data exposure, CRDT
injection                      │
│
Mitigation:
│
│  • TLS 1.3
mandatory                                        │
│  • Certificate
pinning                                      │
│  • HSTS
headers                                             │
```

```
│
│
│  3. MALICIOUS
INSIDER                                        │
│
────────────────────
│
│  Risk: Employee with valid
credentials                      │
│  Impact: Data theft,
sabotage                               │
│
Mitigation:
│
│  • Least privilege
access                                   │
│  • Audit
logging                                            │
│  • Anomaly
detection                                        │
│  • Data access
reviews                                  │
│
│
│  4. TOKEN
THEFT                                               │
│
────────────
│
│  Risk: JWT stolen from device/
memory                        │
│  Impact: Unauthorized sync
operations                       │
│
Mitigation:
│
│  • Short token expiry (15
min)                              │
│  • Secure token storage (Keychain/
Keystore)                 │
│  • Token binding to device
ID                               │
│  • Refresh token
rotation                                   │
│
│
│  5. REPLAY
ATTACK                                            │
│
──────────────
│
│  Risk: Re-send old sync
operations                          │
│  Impact: Data corruption, duplicate
entries                 │
│
Mitigation:
│
│  • CRDT timestamps prevent
duplicates                       │
│  • Operation IDs
(idempotency)                              │
│  • Nonce in
requests                                        │
│
│
└──────────────────────────────────────────────────────────────┘
```

3. Authentication Design

3.1 JWT Token Flow

```
┌──────────────────────────────────────────────────────────────┐
│                    JWT TOKEN
FLOW                            │
├──────────────────────────────────────────────────────────────┤
│
│
│  1.
LOGIN                                                    │
│
────────
│
│  Client ──▶ POST /auth/login ──▶ Auth
Service               │
│             {username,
password,                             │
│              device_id,
device_fingerprint}                  │
│
│
│  Auth
Service:                                               │
│  • Validate
credentials                                      │
│  • Generate access token (15 min
TTL)                       │
│  • Generate refresh token (7 days
TTL)                      │
│  • Bind tokens to
device_id                                 │
│
│
│
Response:
```

```
│
│  {access_token, refresh_token,
expires_in}                  │
│
│
│  2. API
CALLS                                                │
│
──────────
│
│  Client ──▶ Any API ──▶ Middleware validates
JWT            │
│  Header: Authorization: Bearer
<access_token>               │
│
│
│  3. TOKEN
REFRESH                                            │
│
──────────────
│
│  Client ──▶ POST /auth/refresh ──▶ Auth
Service             │
│
{refresh_token}                                  │
│
│
│  Auth
Service:                                               │
│  • Validate refresh
token                                   │
│  • Check device
binding                                     │
│  • Rotate refresh token (old one
invalidated)               │
│  • Issue new access
token                                   │
```

```
│
│
└──────────────────────────────────────────────────────────────┘
```

3.2 JWT Payload

```
{
  "sub": "user_123",
  "vendor_code": "ABC",
  "roles": ["operator"],
  "permissions": ["read:batch", "write:barcode"],
  "assigned_batches": [1, 2, 3],
  "device_id": "device_xyz",
  "iat": 1735113600,
  "exp": 1735114500
}
```

---

4. Authorization Design

4.1 Row-Level Security (PostgreSQL)

```
-- Enable RLS on all barcode tables
ALTER TABLE bc_barcode ENABLE ROW LEVEL SECURITY;
ALTER TABLE bc_parent ENABLE ROW LEVEL SECURITY;
ALTER TABLE bc_batch ENABLE ROW LEVEL SECURITY;

-- Vendor isolation policy
CREATE POLICY vendor_isolation ON bc_barcode
  FOR ALL
  USING (vendor_code =
         current_setting('app.vendor_code')::text);

-- Batch assignment policy
CREATE POLICY batch_access ON bc_barcode
```

```
  FOR SELECT
  USING (
    batch_id IN (
      SELECT batch_id FROM user_batch_assignments
      WHERE user_id = current_setting('app.user_id')::int
    )
  );

-- Write policy (only operators can write)
CREATE POLICY operator_write ON bc_barcode
  FOR INSERT
  WITH CHECK (
    current_setting('app.role')::text = 'operator'
    AND vendor_code =
        current_setting('app.vendor_code')::text
  );
```

4.2 Electric Shape Authorization

```
# Electric configuration
shapes:
  batch_data:
    tables: [bc_parent, bc_barcode, bc_token]
    authorization:
      # JWT claim requirements
      require:
        - claim: vendor_code
          matches: :vendor_code
        - claim: assigned_batches
          contains: :batch_id
      # Write permissions
      write:
        require_role: operator
```

5. Data Encryption

5.1 Encryption at Rest

```
┌──────────────────────────────────────────────────────────────┐
│                 ENCRYPTION AT
REST                             │
├──────────────────────────────────────────────────────────────┤
│
│
│  SERVER
SIDE                                                 │
│
───────────
│
│
PostgreSQL:
│
│  • Disk encryption (LUKS/AWS EBS
encryption)                  │
│  • Column-level encryption for sensitive
fields                  │
│  • Key management via AWS KMS / HashiCorp
Vault                  │
│
│
│  CLIENT
SIDE                                                 │
│
───────────
│
│  SQLite (Mobile/
Desktop):                                  │
│  • SQLCipher
encryption                                     │
│  • Key derived from user PIN + device
key                      │
```

```
│  • Key stored in Keychain (iOS) / Keystore
(Android)        │
│
│
│  IndexedDB
(Web):                                           │
│  • Web Crypto API
encryption                                │
│  • Key derived from password +
PBKDF2                       │
│  • Consider not storing sensitive data in web
version       │
│
│
└────────────────────────────────────────────────────────────┘
```

5.2 Encryption in Transit

```
All connections MUST use TLS 1.3

HTTPS endpoints:
• Certificate: Let's Encrypt / AWS ACM
• HSTS: max-age=31536000; includeSubDomains
• Content-Security-Policy: strict

WebSocket:
• wss:// only (no ws://)
• Same certificate as HTTPS

Mobile certificate pinning:
• Pin to leaf certificate
• Include backup pin for rotation
```

5.3 Field-Level Encryption (Optional)

```
# For extra-sensitive fields (e.g., token values)
from cryptography.fernet import Fernet

class EncryptedField:
    def __init__(self, key: bytes):
        self.fernet = Fernet(key)

    def encrypt(self, value: str) -> str:
        return self.fernet.encrypt(value.encode()).decode()

    def decrypt(self, encrypted: str) -> str:
        return
        self.fernet.decrypt(encrypted.encode()).decode()

# Usage in model
bc_token.token = encrypted_field.encrypt(raw_token)
```

---

6. Security Controls Matrix

Control	Server	Mobile	Web	Priority
TLS 1.3	✅	✅	✅	CRITICAL
JWT auth	✅	✅	✅	CRITICAL
RLS	✅	N/A	N/A	CRITICAL
SQLite encryption	N/A	✅	N/A	HIGH
Certificate pinning	N/A	✅	N/A	HIGH
Rate limiting	✅	N/A	N/A	HIGH

Control	Server	Mobile	Web	Priority
Audit logging	✅	✅	✅	MEDIUM
WAF	✅	N/A	N/A	MEDIUM
Field encryption	Optional	Optional	Optional	LOW

---

7. Incident Response

7.1 Security Incident Types

Type	Severity	Response Time
Data breach	CRITICAL	< 1 hour
Token compromise	HIGH	< 4 hours
DDoS attack	HIGH	< 1 hour
Unauthorized access	MEDIUM	< 24 hours

7.2 Remote Wipe Capability

```
IF device_compromised:
  1. Revoke all tokens for device_id
  2. Add device_id to blocklist
  3. Trigger remote wipe via push notification
  4. Client deletes local SQLite DB
  5. Log incident to security team
```

---

8. Compliance Considerations

Standard	Requirement	Implementation
GDPR	Data minimization	Sync only needed data
GDPR	Right to erasure	Cascade delete in sync
ISO 27001	Access control	RLS + RBAC
SOC 2	Audit trails	All ops logged
PCI DSS	Encryption	TLS + SQLCipher

---

9. Security Checklist

Pre-Production

- □ Penetration testing completed
- □ Security code review done
- □ Dependency vulnerability scan
- □ TLS configuration validated
- □ JWT implementation reviewed
- □ RLS policies tested
- □ Rate limiting configured
- □ WAF rules deployed

Ongoing

- □ Weekly vulnerability scans
- □ Monthly access reviews
- □ Quarterly penetration tests
- □ Annual security audit

SLA & Reliability Design

99.99% Uptime Architecture

---

1. SLA Targets

```
┌──────────────────────────────────────────────────────────────┐
│                    SLA
TARGETS                               │
├──────────────────────────────────────────────────────────────┤
│
│
│  TIER 1: CRITICAL (99.99% = 52 min downtime/
year)           │
│
────────────────────────────────────────────────
│
│  • Barcode scanning (offline-
capable)                       │
│  • Local database
operations                                │
│  • Offline queue
processing                                 │
│
```

```
│
│  TIER 2: HIGH (99.9% = 8.76 hours downtime/
year)                 │
│
───────────────────────────────────────────────
│
│  • Real-time sync
(WebSocket)                               │
│  • Initial sync
(bootstrap)                                 │
│  • API
endpoints                                            │
│
│
│  TIER 3: STANDARD (99.5% = 43.8 hours downtime/
year)             │
│
─────────────────────────────────────────────────
│
│  • Background
sync                                          │
│  • Historical data
access                                   │
│  • Reporting &
analytics                                    │
│
│
└──────────────────────────────────────────────────────────────┘
```

---

2. How Local-First Achieves 99.99%

```
┌──────────────────────────────────────────────────────────────┐
│               LOCAL-FIRST = HIGH
AVAILABILITY                 │
├──────────────────────────────────────────────────────────────┤
```

```
│
│
│  TRADITIONAL
ARCHITECTURE                                    │
│
─────────────────────────
│
│  App ──▶ Network ──▶ Server ──▶
Database                    │
│       │         │
│                               │
│       └─ Failure points
────┘                               │
│
│
│  Availability = 99.9% × 99.9% × 99.9% =
99.7%               │
│
│
│
═══════════════════════════════════════
│
│
│
│  LOCAL-FIRST
ARCHITECTURE                                    │
│
─────────────────────────
│
│  App ──▶ Local SQLite (always
available)                    │
│
│                                                      │
│       └──▶ Background sync (eventually
consistent)          │
│
│
```

```
│  Core operations: 99.99%+ (limited only by app/
device)       │
│  Sync operations: 99.9% (server-
dependent)                      │
│
│
│  User Experience Availability =
99.99%                            │
│
│
└─────────────────────────────────────────────────────────────┘
```

---

3. Failure Mode Analysis

3.1 Failure Scenarios & Mitigations

Failure	Impact	Detection	Mitigation	RTO
Network down	No sync	Connection monitor	Offline mode	0s
Electric crash	No sync	Health check	Auto-restart K8s	30s
PostgreSQL down	No sync	PG health check	Failover to replica	1 min
Client crash	App restart	OS-level	Auto-resume sync	5s
Local DB corrupt	Data loss	Checksum	Re-sync from server	1 min
Full datacenter outage	No sync	Multi-region	DR failover	5 min

3.2 RTO/RPO Targets

```
┌──────────────────────────────────────────────────────────────┐
│                     RTO / RPO
TARGETS                         │
├──────────────────────────────────────────────────────────────┤
│
│
│  RTO (Recovery Time
Objective)                              │
│
─────────────────────────────
│
│  • Local operations: 0 seconds (always
available)             │
│  • Sync resume: < 30 seconds after network
restore            │
│  • Full system recovery: < 5
minutes                        │
│
│
│  RPO (Recovery Point
Objective)                             │
│
─────────────────────────────
│
│  • Local changes: 0 (persisted
immediately)                 │
│  • Server state: < 1 second (real-time
sync)                  │
│  • Worst case (offline period): Duration of
offline          │
│
│
│  Note: With local-first, no data is ever
lost.                  │
│  Data syncs when connection
```

```
restores.                        │
│
│
└──────────────────────────────────────────────────────────────┘
```

---

4. High Availability Architecture

4.1 Server-Side HA

```
┌──────────────────────────────────────────────────────────────┐
│                SERVER-SIDE HA
ARCHITECTURE                   │
├──────────────────────────────────────────────────────────────┤
│
│
│  REGION A (Primary)              REGION B
(DR)               │
│  ──────────────────
────────────────            │
│
│
│  ┌─────────────────┐
┌─────────────────┐         │
│  │ Load Balancer   │────────────│ Load Balancer
│         │
│  │ (Active)        │            │ (Standby)
│         │
│  └────────┬────────┘
└────────┬────────┘         │
│           │
│                   │
│  ┌────────┴────────┐
┌────────┴────────┐         │
│  │ Electric Pod ×3 │            │ Electric Pod ×2
│         │
│   │ (Auto-scaling)  │             │ (Warm standby)
│         │
│   └─────────┬─────────┘
└─────────┬─────────┘          │
│           │
│                   │
│   ┌─────────┴─────────┐
┌─────────┴─────────┐          │
│   │ PostgreSQL      │────────────│ PostgreSQL
│         │
│   │ Primary         │  Streaming │ Replica
│         │
│   │                 │  Replication
│         │
│   └───────────────────┘
└───────────────────┘          │
│
│
│  Failover: Automatic via health
checks                             │
│  DNS: Route53 health-based
routing                                  │
│
│
└──────────────────────────────────────────────────────────────┘
```

4.2 Client-Side Resilience

```
┌──────────────────────────────────────────────────────────────┐
│                CLIENT-SIDE
RESILIENCE                               │
├──────────────────────────────────────────────────────────────┤
│
│
│  1. CONNECTION
MANAGEMENT                                           │
│
```

```
─────────────────────────
│
│  • Automatic reconnection with exponential
backoff          │
│  • Connection pooling for
WebSocket                         │
│  • Fallback from WebSocket → HTTP long-
polling              │
│
│
│  2. DATA
INTEGRITY                                          │
│
─────────────────
│
│  • WAL (Write-Ahead Log) for
SQLite                        │
│  • Transaction support for all
writes                      │
│  • Periodic checksums against
server                       │
│
│
│  3. ERROR
RECOVERY                                           │
│
─────────────────
│
│  • Retry logic for failed sync
operations                  │
│  • Conflict queue for manual
resolution                    │
│  • Automatic re-sync on corruption
detection                 │
│
│
└──────────────────────────────────────────────────────────────┘
```

5. Monitoring & Alerting

5.1 Health Metrics

Metric	Threshold	Alert
Electric pod CPU	> 80%	WARNING
Electric pod memory	> 85%	WARNING
WebSocket connections	> 10K/pod	SCALE UP
Sync latency P99	> 5 seconds	WARNING
Sync error rate	> 1%	CRITICAL
PostgreSQL replication lag	> 10 seconds	CRITICAL
Client offline duration	> 24 hours	INFO

5.2 Alerting Matrix

```
CRITICAL (Page on-call immediately):
• All Electric pods down
• PostgreSQL primary unavailable
• Sync error rate > 5%
• Data integrity checksum mismatch

WARNING (Notify team, investigate):
• Single pod failure
• Sync latency > 30 seconds
• Replication lag > 1 minute
• Memory usage > 85%

INFO (Log, review daily):
• New client version adoption
• Offline client count
• Conflict resolution rate
```

---

6. Disaster Recovery

6.1 DR Strategy

```
┌──────────────────────────────────────────────────────────────┐
│                    DR
STRATEGY                               │
├──────────────────────────────────────────────────────────────┤
│
│
│  TIER 1: AUTOMATIC (Infrastructure
failures)                │
│
────────────────────────────────────────────
│
│  • K8s auto-restart crashed
pods                            │
│  • PostgreSQL auto-failover to
replica                      │
│  • DNS failover to DR
region                                │
│  • No manual intervention
needed                            │
│
│
│  TIER 2: SEMI-AUTOMATIC (Regional
failures)                 │
│
──────────────────────────────────────────
│
│  • Trigger DR
runbook                                       │
```

```
│  • Promote DR PostgreSQL to
primary                         │
│  • Update DNS to point to DR
region                          │
│  • ~5 minute
recovery                                        │
│
│
│  TIER 3: MANUAL (Catastrophic
failures)                       │
│
──────────────────────────────────────
│
│  • Restore from
backup                                      │
│  • Rebuild
infrastructure                                   │
│  • Re-bootstrap clients (data preserved
locally)            │
│  • ~30 minute
recovery                                       │
│
│
│  CLIENT BEHAVIOR DURING
DR:                                      │
│  • Continues operating
offline                                  │
│  • Queues all operations
locally                                │
│  • Auto-syncs when servers
recover                            │
│  • Zero user-facing
downtime                                  │
│
│
└──────────────────────────────────────────────────────────────┘
```

6.2 Backup Strategy

Data	Frequency	Retention	Storage
PostgreSQL full	Daily	30 days	S3 + Glacier
PostgreSQL WAL	Continuous	7 days	S3
Electric state	Hourly	7 days	S3
Client SQLite	On sync	N/A	Server is backup

---

7. Capacity Planning

7.1 Load Estimates

```
Assumptions:
• 100 vendors
• 10,000 active users
• 1,000,000 barcodes per vendor
• 100 scans per user per day

Calculations:
• Total sync connections: 10,000 (peak)
• Writes per second: 1,000,000 scans / 28,800 sec = ~35 writes/sec
• Sync messages per second: 35 × 100 (fanout) = 3,500 msg/sec
```

7.2 Infrastructure Sizing

Component	Size	Scale Trigger
Electric pods	3 × 2 CPU, 4GB RAM	CPU > 70%

Component	Size	Scale Trigger
PostgreSQL	db.r6g.xlarge (4 vCPU, 32GB)	Connections > 500
Redis cache	cache.r6g.large	Memory > 80%
Load Balancer	Application LB	Auto-scales

---

8. SLA Dashboard Metrics

```
┌──────────────────────────────────────────────────────────────┐
│                    SLA
DASHBOARD                                │
├──────────────────────────────────────────────────────────────┤
│
│
│  AVAILABILITY (Last 30
days)                                    │
│
───────────────────────────
│
│  Overall System:     99.95% ███████████████████░░
✓       │
│  Electric Service:   99.98% ████████████████████░
✓       │
│  PostgreSQL:         99.99% ████████████████████░
✓       │
│  Client Operations:  100.0% ████████████████████
✓        │
│
│
│  PERFORMANCE (P99
Latency)                                           │
│
─────────────────────────
```

```
│
│  Initial Sync:        12.3s  ████████░░░░░░░░░░░░
✓        │
│  Incremental Sync:    0.8s   ██░░░░░░░░░░░░░░░░░░
✓        │
│  Local Write:         5ms    ░░░░░░░░░░░░░░░░░░░░
✓        │
│  Conflict Resolution: 50ms   ░░░░░░░░░░░░░░░░░░░░
✓        │
│
│
│
INCIDENTS
│
│
─────────
│
│  This Month:          1 (minor, 5
min)                      │
│  Last Quarter:        3 (total downtime: 18
min)            │
│  SLA Target:          52 min/year
allowed                   │
│  Remaining Budget:    34
min                                 │
│
│
└──────────────────────────────────────────────────────────────┘
```

9. Reliability Checklist

Pre-Production

□ Load testing completed (2× expected peak)

□

Failover testing completed

- □ DR drill executed
- □ Chaos engineering tests passed
- □ Client offline testing (72 hours)
- □ Sync conflict scenarios tested
- □ Monitoring dashboards configured
- □ Alerting rules configured
- □ Runbooks documented

Ongoing

- □ Weekly health checks
- □ Monthly DR drills
- □ Quarterly capacity review
- □ Annual architecture review

Blindspots & Risk Analysis

Edge Cases, Risks, and Mitigation Strategies

1. Risk Overview

```
┌─────────────────────────────────────────────────────────────┐
│                    RISK
MATRIX                               │
├─────────────────────────────────────────────────────────────┤
│
│
│
IMPACT
│
│
▲                                                         │
│
│                                                         │
│  H │  ┌────────┐
┌────────┐                             │
│  I │  │ Clock  │     │ Data
│                             │
│  G │  │ Drift  │     │ Loss
│                             │
│  H │  └────────┘
└────────┘                             │
│
│                                                         │
│  M │  ┌────────┐     ┌────────┐
┌────────┐             │
│  E │  │ Schema │     │Storage │
│Conflict│             │
│  D │  │ Change │     │ Full   │     │ Flood
│             │
│    │  └────────┘     └────────┘
└────────┘             │
│
│                                                         │
│  L │  ┌────────┐
┌────────┐                             │
```

```
│  O │  │ Slow    │     │ User
│                              │
│  W │  │ Sync    │     │ Error
│                              │
│    │  └─────────┘
└─────────┘                              │
│
│                                                  │
│
└─────────────────────────────────────────────▶           │
│       LOW             MEDIUM
HIGH                    │
│
LIKELIHOOD                                 │
│
│
└──────────────────────────────────────────────────────────────┘
```

---

2. Technical Blindspots

2.1 Clock Drift Issue

```
┌──────────────────────────────────────────────────────────────┐
│  BLINDSPOT: Device Clock
Drift                                  │
├──────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
│
│  • CRDT uses timestamps for conflict
resolution              │
│  • If device clock is wrong, wrong data
wins                 │
```

```
│  • Mobile devices can have significantly wrong
clocks        │
│
│
│
SCENARIO:
│
│  Device A clock: 2025-12-25 10:00:00
(correct)              │
│  Device B clock: 2025-12-26 10:00:00 (1 day
ahead!)         │
│
│
│  Both update same barcode → Device B always
wins             │
│  (even for older
changes)                                    │
│
│
│  IMPACT:
HIGH                                                    │
│  LIKELIHOOD:
MEDIUM                                             │
│
│
│
MITIGATION:
│
│  1. Use Hybrid Logical Clock (HLC) instead of wall
clock    │
│  2. Validate device time delta on
sync                        │
│  3. Warn user if clock is significantly
off                   │
│  4. Server can reject ops with unrealistic
timestamps        │
│
```

```
│
│
IMPLEMENTATION:
│
│  • On sync connect, compare device time vs server
time        │
│  • If |delta| > 5 minutes, warn
user                             │
│  • If |delta| > 1 hour, block sync until
fixed                   │
│
│
└──────────────────────────────────────────────────────────────┘
```

2.2 Storage Exhaustion

```
┌──────────────────────────────────────────────────────────────┐
│  BLINDSPOT: Client Storage
Full                              │
├──────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
│
│  • Mobile devices have limited
storage                           │
│  • SQLite DB can grow large with many
barcodes                        │
│  • System may prevent writes when storage is
low               │
│
│
│
SCENARIO:
│
│  • User syncs large batch (100K
barcodes)                   │
│  • Phone storage at
95%                                     │
│  • SQLite write fails → sync
stuck                          │
│  • User cannot scan new
barcodes                            │
│
│
│  IMPACT:
HIGH                                                │
│  LIKELIHOOD:
MEDIUM                                          │
│
│
│
MITIGATION:
│
│  1. Monitor storage before
sync                              │
│  2. Implement LRU eviction for old batch
data               │
│  3. Compress local DB periodically
(VACUUM)                 │
│  4. Warn user when storage < 500MB
free                     │
│  5. Allow partial sync (critical data
only)                 │
│
│
│  STORAGE
ESTIMATION:                                         │
│  • 100K barcodes × 500 bytes = 50
MB                        │
│  • + indexes + overhead = ~100 MB per
batch                 │
│  • Reserved minimum: 200 MB for
```

```
operations                  │
│
│
└──────────────────────────────────────────────────────────────┘
```

2.3 Schema Migration

```
┌──────────────────────────────────────────────────────────────┐
│  BLINDSPOT: Schema Changes During Active
Sync                │
├──────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
│
│  • Server schema changes (new column, new
table)            │
│  • Client has old schema in
SQLite                          │
│  • Sync breaks or data is
lost                              │
│
│
│
SCENARIO:
│
│  1. Server adds new column:
bc_barcode.quality_score        │
│  2. Old client tries to
sync                                │
│  3. Client receives row with unknown
column                 │
│  4. Insert fails or column is silently
dropped              │
│
│
```

```
│  IMPACT:
HIGH                                                │
│  LIKELIHOOD: LOW (planning reduces
this)                    │
│
│
│
MITIGATION:
│
│  1. Version the sync
protocol                               │
│  2. Client reports its schema version on
connect             │
│  3. Server downgrades response for old
clients              │
│  4. Force app update for breaking schema
changes            │
│  5. Use backward-compatible migrations
only                 │
│
│
│  MIGRATION
RULES:                                            │
│  ✅ ADD column with default
value                           │
│  ✅ ADD new
table                                           │
│  ⚠️ RENAME column (requires version
gate)                  │
│  ❌ DROP column (never, use is_deprecated
flag)             │
│  ❌ CHANGE column type
(never)                              │
│
│
└─────────────────────────────────────────────────────────────┘
```

2.4 Conflict Flood

```
┌────────────────────────────────────────────────────────────┐
│  BLINDSPOT: Mass Conflict During
Reconnect                   │
├────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
│
│  • Many users offline simultaneously (network
outage)       │
│  • All make changes to same
data                            │
│  • All reconnect at same
time                               │
│  • Massive conflict resolution
load                         │
│
│
│
SCENARIO:
│
│  • Factory WiFi down for 4
hours                           │
│  • 50 operators scanning barcodes
offline                  │
│  • WiFi restored → all sync at
once                         │
│  • 50,000 CRDT operations to
merge                          │
│  • Server overwhelmed, sync
fails                           │
│
│
│  IMPACT:
```

```
MEDIUM                                                     │
│  LIKELIHOOD:
MEDIUM                                                 │
│
│
│
MITIGATION:
│
│  1. Jitter reconnection (random delay 0-30
sec)                │
│  2. Rate limit sync operations per
client                      │
│  3. Priority queue (older offline clients
first)               │
│  4. Circuit breaker on Electric
service                        │
│  5. Auto-scale pods on connection
spike                        │
│
│
│
IMPLEMENTATION:
│
│  reconnect_delay = random(0, 30) + (offline_duration /
100) │
│
│
└──────────────────────────────────────────────────────────────┘
```

---

3. Business Logic Blindspots

3.1 Double Pairing Prevention

```
┌──────────────────────────────────────────────────────────────┐
│  BLINDSPOT: Barcode Paired to Multiple
```

```
Parents                │
├──────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
│
│  • Business rule: 1 barcode = 1 parent
only                  │
│  • Two users offline pair same barcode to different
parents │
│  • CRDT merges both → barcode has 2
parents                  │
│  • Inventory count
incorrect                                │
│
│
│  IMPACT: HIGH (business data
integrity)                         │
│  LIKELIHOOD:
MEDIUM                                              │
│
│
│
MITIGATION:
│
│  1. First-write-wins for pairing (not
LWW)                   │
│  2. Server validates pairing on
sync                         │
│  3. Reject second pairing, notify
user                       │
│  4. Add "pairing conflict" queue for manual
resolution      │
│
│
│
```

```
DETECTION:
│
│  • CHECK constraint: bc_barcode.parent_id is unique per
row │
│  • Periodic audit: SELECT barcodes with >1 pairing
log      │
│
│
└──────────────────────────────────────────────────────────────┘
```

3.2 Deleted Data Resurrection

```
┌──────────────────────────────────────────────────────────────┐
│  BLINDSPOT: Deleted Items Coming
Back                            │
├──────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
│
│  • Admin deletes barcode on
server                              │
│  • Offline user edits same
barcode                              │
│  • User syncs → barcode
"resurrects"                           │
│
│
│
SCENARIO:
│
│  T1: Admin sets is_deleted = true
(server)                      │
│  T2: Offline user updates note (local, timestamp after
T1)   │
│  T3: User syncs → LWW picks user's update (T2 >
T1)          │
│  T4: is_deleted reverted to false!
💀                        │
│
│
│  IMPACT:
MEDIUM                                              │
│  LIKELIHOOD:
LOW                                             │
│
│
│
MITIGATION:
│
│  1. Use tombstone semantics (delete =
permanent)              │
│  2. is_deleted uses OR-set (true always
wins)                   │
│  3. Separate delete operation from
update                     │
│  4. Server rejects updates to deleted
rows                    │
│
│
│  CRDT
RULE:                                                  │
│  is_deleted field uses "add-wins"
semantics:                │
│  DELETE(T1) + UPDATE(T2) = DELETED (regardless of T1 vs
T2) │
│
│
└──────────────────────────────────────────────────────────────┘
```

3.3 Stale Read Issues

```
┌────────────────────────────────────────────────────────────────
│  BLINDSPOT: Acting on Stale
Data                            │
├────────────────────────────────────────────────────────────────
│
│
│
PROBLEM:
│
│  • User views data that's
outdated                         │
│  • Makes decision based on stale
info                          │
│  • Sync updates show different
reality                         │
│
│
│
SCENARIO:
│
│  • User A sees: "Parent X has 5 barcodes
paired"            │
│  • Actually server shows: "Parent X has 10
barcodes"         │
│  • User A pairs 5 more (thinks they're completing
it)        │
│  • Sync shows Parent X now has 15
(overfilled!)              │
│
│
│  IMPACT:
MEDIUM                                                     │
│  LIKELIHOOD:
MEDIUM                                                 │
│
```

```
│
│
MITIGATION:
│
│  1. Show "last synced" timestamp
prominently                │
│  2. Warn user if data is > 5 min
stale                      │
│  3. For critical operations, require fresh
sync             │
│  4. Use optimistic locking with version
check               │
│
│
│  UX
RECOMMENDATION:                                  │
│  "Data from 2 hours ago" warning
banner                      │
│  "Sync now" button for critical
screens                     │
│
│
└──────────────────────────────────────────────────────────────┘
```

---

4. Operational Blindspots

4.1 Long Offline Period

```
┌──────────────────────────────────────────────────────────────┐
│  BLINDSPOT: Client Offline for
Weeks                        │
├──────────────────────────────────────────────────────────────┤
│
│
│
```

```
PROBLEM:
│
│  • Client offline for extended period
(weeks)               │
│  • Massive accumulated delta on
server                      │
│  • Schema may have
changed                                   │
│  • Initial re-sync takes very long or
fails                 │
│
│
│  IMPACT:
MEDIUM                                              │
│  LIKELIHOOD:
LOW                                             │
│
│
│
MITIGATION:
│
│  1. Track last sync timestamp per
device                    │
│  2. If > 7 days offline, force full re-
sync                 │
│  3. Warn user before sync: "Large sync
required"            │
│  4. Allow background sync with
progress                     │
│  5. Preserve local changes during re-
sync                   │
│
│
│  THRESHOLD
LOGIC:                                            │
│  if (offline_duration < 7
days):                            │
```

```
│
incremental_sync()                                       │
│
else:
│
│      warn_user("Large sync
required")                         │
│
full_resync_with_merge()                                 │
│
│
└─────────────────────────────────────────────────────────────┘
```

4.2 Electric Service Memory Leak

```
┌─────────────────────────────────────────────────────────────┐
│  BLINDSPOT: Memory Leak in Long-Running
Service             │
├─────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
│
│  • Electric service runs
24/7                              │
│  • Memory slowly increases over days/
weeks                    │
│  • Eventually OOM kills the
service                             │
│  • Clients lose sync
connection                              │
│
│
│  IMPACT: HIGH (service
outage)                                │
│  LIKELIHOOD:
```

```
LOW                                             │
│
│
│
MITIGATION:
│
│  1. Monitor memory usage over
time                          │
│  2. Set K8s memory limits (hard
cap)                        │
│  3. Automatic pod rolling restart
weekly                    │
│  4. Alert on memory growth
trend                            │
│  5. Profile service under
load                              │
│
│
│  K8s
CONFIG:                                                 │
│
resources:
│
│
limits:                                                   │
│      memory:
4Gi                                             │
│
requests:                                                 │
│      memory:
2Gi                                             │
│
│
└────────────────────────────────────────────────────────────┘
```

5. User Experience Blindspots

5.1 Sync Progress UX

```
┌──────────────────────────────────────────────────────────────┐
│  BLINDSPOT: User Doesn't Know Sync
State                     │
├──────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
│
│  • User doesn't know if they're synced or
not               │
│  • Makes changes thinking they're saved to
server           │
│  • Logs out or uninstalls
app                               │
│  • Changes lost (were still in local
queue)                 │
│
│
│  IMPACT: HIGH (user trust, data
loss)                       │
│  LIKELIHOOD:
MEDIUM                                          │
│
│
│
MITIGATION:
│
│  1. Always show sync status
indicator                       │
│  2. Show pending changes
count                              │
│  3. Warn before logout if pending changes
```

```
exist                 │
│  4. Block uninstall if pending changes (if
possible)           │
│  5. Regular "all synced"
notification                          │
│
│
│  UX
ELEMENTS:                                                       │
│
┌───────────────────────────┐
│
│  │ ✓ All synced            │
(green)                          │
│  │ ↻ Syncing 5 items...    │  (blue,
animated)               │
│  │ ⚠ 3 pending changes     │
(yellow)                         │
│  │ ✗ Offline - 12 pending  │
(red)                            │
│
└───────────────────────────┘
│
│
│
└──────────────────────────────────────────────────────────────┘
```

5.2 Conflict Notification

```
┌──────────────────────────────────────────────────────────────┐
│  BLINDSPOT: User Unaware of Auto-Resolved
Conflicts             │
├──────────────────────────────────────────────────────────────┤
│
│
│
PROBLEM:
```

```
│
│  • CRDT auto-resolves
conflicts                              │
│  • User's changes might be "lost" (LWW picked
other)        │
│  • User doesn't know their change didn't
win                   │
│  • Confusion when data differs from what they
entered       │
│
│
│  IMPACT: MEDIUM (user
confusion)                             │
│  LIKELIHOOD:
MEDIUM                                           │
│
│
│
MITIGATION:
│
│  1. Log all conflict
resolutions                             │
│  2. Notify user when their change was
superseded            │
│  3. Show conflict history for
debugging                    │
│  4. For critical fields, require manual
resolution          │
│
│
│
NOTIFICATION:
│
│  "Your note was updated by another user (John, 5 min
ago).  │
│   Your version: 'QC
OK'                                       │
```

```
│   Current version: 'QC Failed - see
supervisor'"              │
│
│
└──────────────────────────────────────────────────────────┘
```

6. Risk Mitigation Summary

Blindspot	Severity	Mitigation Status
Clock Drift	HIGH	⚠️ Need HLC implementation
Storage Full	HIGH	⚠️ Need LRU eviction
Schema Migration	HIGH	✅ Version protocol
Conflict Flood	MEDIUM	⚠️ Need jitter + rate limit
Double Pairing	HIGH	⚠️ Need server validation
Deleted Resurrection	MEDIUM	✅ OR-set semantics
Stale Read	MEDIUM	⚠️ Need staleness indicator
Long Offline	MEDIUM	✅ Force re-sync logic
Memory Leak	HIGH	✅ K8s limits + restart
Sync State UX	HIGH	⚠️ Need status indicator
Conflict UX	MEDIUM	⚠️ Need notifications

7. Recommended Pre-Launch Checklist

Critical (Block Launch)

- □ HLC implementation for timestamps
- □ Server-side pairing validation
- □ Sync status indicator in UI
- □ Storage monitoring + warnings
- □ Reconnection jitter algorithm

High Priority (Launch Within Week)

- □ LRU eviction for old batch data
- □ Conflict notification system
- □ Staleness warning UI
- □ Memory monitoring dashboard

Nice to Have

- □ Conflict history viewer
- □ Advanced analytics on sync patterns
- □ Custom conflict resolution UI

User Analysis

Impact on End Users & UX Considerations

1. User Personas

1.1 Primary Users

```
┌─────────────────────────────────────────────────────────────┐
│                    USER
PERSONAS                             │
├─────────────────────────────────────────────────────────────┤
│
│
│  PERSONA 1: Factory
Operator                                 │
│
──────────────────────────
│
│  Name:
Budi                                              │
│  Role: Production line barcode
scanner                       │
│  Tech Level: Basic smartphone
user                           │
│  Environment: Factory floor, sometimes weak
WiFi             │
│  Pain
Points:                                           │
│  • App freezes when scanning
quickly                         │
│  • Loses work when WiFi
drops                            │
```

```
│  • Confused by error
messages                               │
│
Goals:
│
│  • Scan barcodes quickly without
waiting                    │
│  • Not lose work when connection is
unstable                │
│
│
│
─────────────────────────────────────────────
│
│
│
│  PERSONA 2: Warehouse
Supervisor                             │
│
────────────────────────────
│
│  Name:
Dewi                                                  │
│  Role: Manages inbound/outbound, oversees
operators          │
│  Tech Level:
Intermediate                                    │
│  Environment: Warehouse, moves between WiFi
zones                 │
│  Pain
Points:                                                │
│  • Can't see real-time status from
operators                │
│  • Data conflicts between team
members                        │
│  • Reports show stale
data                                  │
```

```
│
Goals:
│
│  • Real-time visibility of
operations                       │
│  • Quick conflict
resolution                                │
│  • Accurate
reporting                                      │
│
│
│
─────────────────────────────────────────────
│
│
│
│  PERSONA 3: Admin /
Manager                                  │
│
─────────────────────────
│
│  Name:
Andi                                                  │
│  Role: System admin, batch
management                        │
│  Tech Level:
Advanced                                       │
│  Environment: Office, stable
connection                      │
│  Pain
Points:                                                │
│  • Managing multiple batches across
vendors                 │
│  • Understanding sync status across
devices                 │
│  • Debugging issues from
field                              │
```

```
│
Goals:
│
│   • Dashboard for all sync
statuses                                  │
│   • Ability to force sync on
devices                                 │
│   • Audit trail for all
operations                                 │
│
│
└──────────────────────────────────────────────────────────────┘
```

2. User Journey: Before vs After

2.1 Current State (Online-Only)

```
┌──────────────────────────────────────────────────────────────┐
│             CURRENT USER JOURNEY (ONLINE-
ONLY)                 │
├──────────────────────────────────────────────────────────────┤
│
│
│  Budi starts
shift                                           │
│
│                                                        │
│
▼                                                        │
│  Opens app, waits for data to load ⏳ (30
seconds)                  │
│
│                                                        │
│
▼                                                        │
```

```
│  Starts scanning
barcodes                                    │
│
│                                                        │
│
▼                                                       │
│  WiFi drops 📶
❌                                              │
│
│                                                        │
│
▼                                                       │
│  ❌ App shows error: "No
connection"                          │
│  ❌ Cannot scan
barcodes                                     │
│  ❌ Budi waits,
frustrated                                   │
│
│                                                        │
│
▼                                                       │
│  WiFi returns (5 min
later)                                   │
│
│                                                        │
│
▼                                                       │
│  ❌ App needs to reload all data
again                        │
│  ❌ 5 minutes of productive time
lost                         │
│
│                                                        │
│
▼                                                       │
│  Continues
```

```
scanning                                          │
│
│                                                       │
│
▼                                                       │
│  End of shift: Some scans may have been
lost                    │
│
│
│  TOTAL DOWNTIME: ~30 min/day per
operator                       │
│
│
└─────────────────────────────────────────────────────────────┘
```

2.2 Future State (Local-First)

```
┌─────────────────────────────────────────────────────────────┐
│             FUTURE USER JOURNEY (LOCAL-
FIRST)                     │
├─────────────────────────────────────────────────────────────┤
│
│
│  Budi starts
shift                                           │
│
│                                                       │
│
▼                                                       │
│  Opens app, data already available ✅
(instant)                      │
│  (synced in background since last
session)                           │
│
│                                                       │
│
▼                                                       │
```

```
│  Starts scanning
barcodes                                     │
│  Each scan: instant feedback
✅                               │
│
│                                                        │
│
▼                                                        │
│  WiFi drops 📶
❌                                               │
│
│                                                        │
│
▼                                                        │
│  ✅ App shows: "Offline mode - changes will sync
later"      │
│  ✅ Budi continues scanning without
interruption             │
│  ✅ All scans saved to local
database                         │
│
│                                                        │
│
▼                                                        │
│  WiFi returns (5 min
later)                                   │
│
│                                                        │
│
▼                                                        │
│  ✅ App background syncs changes (Budi doesn't
notice)       │
│  ✅ "12 items synced"
notification                            │
│
│                                                        │
│
```

```
▼                                                                      │
│  Continues scanning
seamlessly                                         │
│
│                                                                      │
│
▼                                                                      │
│  End of shift: All data synced, zero loss
✅                    │
│
│
│  TOTAL DOWNTIME: ~0 min/day per
operator                             │
│
│
└──────────────────────────────────────────────────────────────────────────┘
```

3. UX Design Recommendations

3.1 Sync Status Indicator

```
┌──────────────────────────────────────────────────────────────────────────┐
│                    SYNC STATUS UI
DESIGNS                                │
├──────────────────────────────────────────────────────────────────────────┤
│
│
│  OPTION A: Status Bar
(Recommended)                                  │
│
───────────────────────────────────────────
│
│
┌─────────────────────────────────────────────────────┐
│
```

```
│  │ ≡  Barcode Scanner              ✓ Synced
│               │
│
├───────────────────────────────────────────┤
│
│  │
│               │
│  │        [Main content area]
│               │
│  │
│               │
│
└───────────────────────────────────────────┘
│
│
│
│
States:
│
│  ✓ Synced
(green)                                          │
│  ↻ Syncing... (blue,
animated)                             │
│  ⚠ 3 pending
(yellow)                                      │
│  ✗ Offline
(red)                                           │
│
│
│
─────────────────────────────────────────────
│
│
│
│  OPTION B: Floating
Badge                                   │
│
```

```
────────────────────────────────
│
│
┌──────────────────────────────────────────────────────┐
│
│   │
│                   │
│   │         [Main content area]
│                   │
│   │
│                   │
│   │                                                            ┌────┐
│                   │
│   │                                                            │ 3 │ │   ←
pending       │
│   │                                                            └────┘ │
badge         │
│
└──────────────────────────────────────────────────────┘
│
│
│
└─────────────────────────────────────────────────────────────────────────────┘
```

3.2 Offline Mode Banner

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                     OFFLINE MODE
BANNER                                     │
├─────────────────────────────────────────────────────────────────────────────┤
│
│
│
┌──────────────────────────────────────────────────────┐
│
│   │ 📶 You're offline
│                   │
│  │ Your changes will sync when connected
│               │
│
└───────────────────────────────────────────┘
│
│
│
│
Behavior:
│
│  • Slides down when connection
lost                         │
│  • Background: muted orange
(#FFF3CD)                       │
│  • Auto-dismisses when back
online                          │
│  • "Reconnecting..." state while
attempting                 │
│
│
│
─────────────────────────────────────────────
│
│
│
│  Back online
notification:                                   │
│
┌───────────────────────────────────────────┐
│
│  │ ✓ Back online! 12 items synced
│               │
│
└───────────────────────────────────────────┘
│
│
```

```
│
└──────────────────────────────────────────────────────────────────────────┘
```

3.3 Conflict Resolution UI

```
┌──────────────────────────────────────────────────────────────────────────┐
│                CONFLICT RESOLUTION
UI                           │
├──────────────────────────────────────────────────────────────────────────┤
│
│
│  For auto-resolved conflicts (notification
only):              │
│
│
│
┌───────────────────────────────────────────┐
│
│  │ ⚠ Update conflict resolved
│              │
│  │ Barcode ABC123 was updated by John
│              │
│  │ Your change: "QC OK"
│              │
│  │ Applied change: "QC Failed"
│              │
│  │                             [View] [OK]
│              │
│
└───────────────────────────────────────────┘
│
│
│
│
─────────────────────────────────────────────
│
│
```

```
│
│  For manual resolution (rare, critical
conflicts):           │
│
│
│
┌──────────────────────────────────────────┐
│
│  │ ⚠ Pairing conflict
│            │
│  │
│            │
│  │ Barcode ABC123 was paired to:
│            │
│  │
│            │
│  │ ○ Parent-001 (by you, 10:30 AM)
│            │
│  │ ○ Parent-002 (by John, 10:32 AM)
│            │
│  │
│            │
│  │ [Ask Supervisor] [Keep Mine] [Keep John]
│            │
│
└──────────────────────────────────────────┘
│
│
│
└────────────────────────────────────────────────────────────┘
```

4. User Communication Plan

4.1 Training Materials

Material	Audience	Format	Duration
Quick Start Guide	All users	PDF/Video	5 min
Offline Mode Training	Operators	Video	10 min
Conflict Resolution	Supervisors	Video + Quiz	15 min
Admin Dashboard	Admins	Live training	1 hour

4.2 Key Messages

```
┌─────────────────────────────────────────────────────────────┐
│                 KEY USER
MESSAGES                             │
├─────────────────────────────────────────────────────────────┤
│
│
│  MESSAGE 1: "Your work is always
saved"                        │
│
─────────────────────────────────────
│
│  "Even without internet, every scan you make is
saved       │
│   on your device. When internet returns, it
syncs           │
│   automatically. You never lose
work."                        │
│
│
│  MESSAGE 2: "Keep working, we handle the
rest"                 │
```

```
│
──────────────────────────────────────────
│
│  "See the offline icon? Don't worry! Just keep
scanning.    │
│   The app will sync everything when
connected."             │
│
│
│  MESSAGE 3: "Conflicts are rare, but we've got
you"          │
│
────────────────────────────────────────────────
│
│  "If two people edit the same thing, we pick the
latest.    │
│   You'll see a notification if your change was
replaced."   │
│
│
│  MESSAGE 4: "Check the sync
status"                          │
│
─────────────────────────────────
│
│  "The icon at the top shows sync
status:                     │
│   ✓ = All good, ⚠ = Some pending, ✗ =
Offline"              │
│
│
└──────────────────────────────────────────────────────────────┘
```

5. User Feedback Mechanisms

5.1 In-App Feedback

```
Trigger points for feedback collection:

1. After first week of use
   → "How's the new offline mode working for you?"
   → Rating 1-5 + optional comment

2. After conflict resolution
   → "Was this conflict easy to understand?"
   → Yes/No + optional comment

3. After large sync (>1000 items)
   → "How was the sync experience?"
   → Rating 1-5

4. Error recovery
   → "Did the app recover correctly?"
   → Yes/No
```

5.2 Analytics to Track

Metric	Purpose	Target
Offline session duration	Understand offline patterns	Track avg
Sync success rate	Reliability	> 99.9%
Conflict frequency	Workflow issues	< 1% of ops
Pending items at logout	User awareness	< 10 avg
Time to first scan	App startup perf	< 3 sec

6. Rollout Strategy

6.1 Phased Rollout

```
┌──────────────────────────────────────────────────────────────┐
│                  ROLLOUT
PHASES                                     │
├──────────────────────────────────────────────────────────────┤
│
│
│  PHASE 1: Internal Testing (Week
1-2)                            │
│
─────────────────────────────────
│
│  • 5 internal
users                                             │
│  • All features
enabled                                         │
│  • Direct feedback
channel                                      │
│  • Fix critical
issues                                          │
│
│
│  PHASE 2: Beta Vendor (Week
3-4)                                │
│
─────────────────────────────
│
│  • 1 vendor (50
users)                                          │
│  • On-site
training                                             │
│  • Daily check-
ins                                              │
│  • Gather usage
```

```
patterns                                    │
│
│
│  PHASE 3: Expanded Beta (Week
5-6)                          │
│
─────────────────────────────────
│
│  • 3 more vendors (150
users)                               │
│  • Remote
training                                          │
│  • Weekly feedback
sessions                                 │
│
│
│  PHASE 4: General Availability (Week
7+)                    │
│
──────────────────────────────────────
│
│  • All
vendors                                              │
│  • Self-serve training
materials                            │
│  • Normal support
channels                                  │
│
│
│  ROLLBACK
PLAN:                                              │
│  • Feature flag to disable offline
mode                     │
│  • Revert to online-only if critical
issues                 │
│  • Data preserved in both
modes                             │
```

```
│
│
└────────────────────────────────────────────────────────────────────┘
```

6.2 Success Criteria per Phase

Phase	Criteria	Threshold
Phase 1	No data loss	100%
Phase 1	Sync success	> 95%
Phase 2	User satisfaction	> 4/5
Phase 2	Downtime related tickets	-50%
Phase 3	Conflict resolution rate	> 99% auto
Phase 4	Adoption rate	> 90% active users

---

7. Support Considerations

7.1 New Support Scenarios

Scenario	User Says	Resolution
Pending items stuck	“It says 5 pending for hours”	Check connectivity, force sync
Data mismatch	“My scan is missing”	Check conflict log, verify sync
Slow sync	“Sync takes forever”	Check data volume, network
Can’t pair	“Barcode won’t pair”	

Scenario	User Says	Resolution
		Check if already paired (conflict)

7.2 Support Tools Needed

□ Admin dashboard with device sync status

□ Ability to view pending items per device

□ Force sync trigger for specific device

□ Conflict log viewer

□ Device sync history

---

8. Expected Outcomes

8.1 Quantitative Benefits

Metric	Before	After	Improvement
Daily downtime/user	30 min	~0 min	-100%
Scan success rate	95%	99.9%	+5%
Data loss incidents	2/month	0/month	-100%
Support tickets (sync)	50/week	10/week	-80%

8.2 Qualitative Benefits

- ✅ Reduced user frustration
- ✅ Higher confidence in system reliability
- ✅ Faster onboarding (app works anywhere)

- ✅ Better field operation flexibility
- ✅ Improved data accuracy

Implementation Plan

Phased Rollout for ElectricSQL Local-First Sync

---

1. Project Timeline

```
┌──────────────────────────────────────────────────────────────┐
│                    PROJECT
TIMELINE                           │
├──────────────────────────────────────────────────────────────┤
│
│
│  PHASE 1: FOUNDATION (Week
1-2)                              │
│  ├── Infrastructure
setup                                      │
│  ├── ElectricSQL
deployment                                     │
│  └── Database
preparation                                       │
│
│
│  PHASE 2: CORE SYNC (Week
3-4)                               │
│  ├── Shape
definitions                                         │
│  ├── Client library
integration                                │
│  └── Basic sync
functionality                                 │
```

```
│
│
│  PHASE 3: OFFLINE CAPABILITY (Week
5-6)                         │
│  ├── Offline queue
implementation                              │
│  ├── Conflict resolution
logic                              │
│  └── UI sync
indicators                                      │
│
│
│  PHASE 4: TESTING & HARDENING (Week
7-8)                        │
│  ├── Load
testing                                          │
│  ├── Chaos
engineering                                     │
│  └── Security
audit                                          │
│
│
│  PHASE 5: ROLLOUT (Week
9-12)                                    │
│  ├── Beta
testing                                          │
│  ├── User
training                                         │
│  └── General
availability                                   │
│
│
│  TOTAL: 12
weeks                                              │
│
│
└──────────────────────────────────────────────────────────────┘
```

2. Phase 1: Foundation (Week 1-2)

2.1 Infrastructure Tasks

Task	Owner	Duration	Dependencies
Deploy PostgreSQL with logical replication	DevOps	2 days	None
Deploy ElectricSQL service	DevOps	2 days	PostgreSQL
Configure load balancer	DevOps	1 day	ElectricSQL
Set up monitoring (Prometheus/ Grafana)	DevOps	1 day	ElectricSQL
Configure TLS certificates	DevOps	1 day	Load balancer

2.2 Database Preparation

```sql
-- Enable logical replication (postgresql.conf)
wal_level = logical

-- Create publication for Electric
CREATE PUBLICATION electric_pub FOR TABLE
  app_barcode.bc_batch,
  app_barcode.bc_parent,
  app_barcode.bc_barcode,
  app_barcode.bc_token,
  app_barcode.bc_config,
  app_barcode.bc_pair_brcdxparent;

-- Enable RLS on sync tables
ALTER TABLE app_barcode.bc_barcode ENABLE ROW LEVEL SECURITY;
ALTER TABLE app_barcode.bc_parent ENABLE ROW LEVEL SECURITY;

-- Create RLS policies
CREATE POLICY vendor_isolation ON app_barcode.bc_barcode
  USING (vendor_code = current_setting('app.vendor_code')::text);
```

2.3 Deliverables

- □ PostgreSQL deployed with WAL level = logical
- □ ElectricSQL running and connected to PostgreSQL
- □ HTTPS endpoint accessible
- □ Basic health monitoring in place
- □ RLS policies created

---

3. Phase 2: Core Sync (Week 3-4)

3.1 Backend Tasks

Task	Owner	Duration
Define shape configurations	Backend	2 days

Task	Owner	Duration
Implement JWT auth for Electric	Backend	2 days
Create sync status API endpoints	Backend	1 day
Write shape authorization rules	Backend	2 days

3.2 Shape Definitions

```yaml
# electric-config.yaml
shapes:
  vendor_master:
    tables:
      - bc_batch
      - bc_config
    where: "vendor_code = :vendor_code AND is_deleted = false"

  batch_data:
    tables:
      - bc_parent
      - bc_barcode
      - bc_token
    where: "batch_id = :batch_id AND is_deleted = false"

  pairing_logs:
    tables:
      - bc_pair_brcdxparent
    where: "vendor_code = :vendor_code AND created_at > now() - interval '7 days'"
```

3.3 Client Integration

Task	Owner	Duration
Add electric-sql client library	Frontend	1 day

Task	Owner	Duration
Implement SQLite local storage	Frontend	2 days
Create sync service wrapper	Frontend	2 days
Integrate with existing UI	Frontend	3 days

3.4 Deliverables

- □ Shapes defined and tested
- □ JWT auth working with Electric
- □ Client can sync data from server
- □ Data visible in local SQLite
- □ Basic CRUD operations working

---

4. Phase 3: Offline Capability (Week 5-6)

4.1 Offline Queue

Task	Owner	Duration
Implement pending_operations table	Frontend	1 day
Create write interceptor	Frontend	2 days
Implement queue processor	Frontend	2 days
Add retry logic with backoff	Frontend	1 day

4.2 Conflict Resolution

Task	Owner	Duration
Implement CRDT merge for barcode	Backend/Frontend	2 days
Add first-write-wins for pairing	Backend	1 day
Create conflict notification system	Frontend	2 days
Server-side validation for conflicts	Backend	2 days

4.3 UI Components

Task	Owner	Duration
Sync status indicator	Frontend	1 day
Offline mode banner	Frontend	1 day
Pending items badge	Frontend	1 day
Conflict resolution modal	Frontend	2 days

4.4 Deliverables

- □ Offline writes queued and processed
- □ Conflicts auto-resolved via CRDT
- □ Manual resolution UI for edge cases
- □ Clear sync status in UI
- □ Graceful offline/online transitions

5. Phase 4: Testing & Hardening (Week 7-8)

5.1 Load Testing

Test	Tool	Target
Concurrent connections	k6	10,000 connections
Sync throughput	k6	1,000 ops/sec
Initial sync time	Custom	< 30 sec for 100K rows
Reconnection storm	Custom	1,000 simultaneous reconnects

5.2 Chaos Engineering

Test	Method	Expected Outcome
Network partition	tc netem	Client continues offline
Electric pod crash	kubectl delete	K8s restarts, clients reconnect
PostgreSQL failover	pg_ctl stop	Replica promoted, no data loss
Full sync after 7 days offline	Manual	Complete resync successful

5.3 Security Audit

- □ Penetration testing
- □ JWT implementation review
- □

RLS policy verification

□ Data exposure analysis

□ Dependency vulnerability scan

5.4 Deliverables

□ Load test report with recommendations

□ Chaos test results documented

□ Security audit passed

□ Performance optimizations applied

□ Runbooks created for incident response

---

6. Phase 5: Rollout (Week 9-12)

6.1 Beta Testing (Week 9-10)

Week	Scope	Users
Week 9	Internal team	5
Week 10	Single vendor	50

6.2 Training (Week 11)

Material	Audience	Delivery
Quick Start Guide	All users	Self-serve
Offline Mode Video	Operators	Async

Material	Audience	Delivery
Admin Training	Admins	Live session
Support Runbook	Support team	Classroom

6.3 General Availability (Week 12)

- □ Feature flag enabled for all vendors
- □ Monitoring dashboards active
- □ Support team trained
- □ Rollback plan tested
- □ Success metrics tracking

---

7. Resource Requirements

7.1 Team

Role	FTE	Duration
Backend Developer	1.5	12 weeks
Frontend Developer	1.5	10 weeks
DevOps Engineer	0.5	12 weeks
QA Engineer	1.0	6 weeks
Product Manager	0.5	12 weeks

7.2 Infrastructure Cost

Component	Monthly Cost
Electric pods (3x)	$300
PostgreSQL (RDS)	$400
Load Balancer	$50
Monitoring	$100
Total	**$850/month**

---

8. Risk Mitigation

Risk	Likelihood	Impact	Mitigation
Electric instability	Medium	High	Fallback to REST API
Performance issues	Medium	Medium	Progressive rollout
User resistance	Low	Medium	Training + champions
Data migration issues	Low	High	Keep both systems parallel

---

9. Success Metrics

Metric	Baseline	Target	Measurement
Offline availability	0%	100%	Feature works offline
Sync success rate	N/A	99.9%	Monitoring

Metric	Baseline	Target	Measurement
User satisfaction	N/A	4.5/5	Survey
Support tickets (sync)	50/week	10/week	Ticket tracking
Data loss incidents	2/month	0/month	Incident reports

---

10. Checklist for Go-Live

Technical Readiness

- □ All phases completed
- □ Load testing passed
- □ Security audit passed
- □ Monitoring configured
- □ Alerting configured
- □ Runbooks documented
- □ Rollback tested

Business Readiness

- □ Training materials ready
- □ Support team trained
- □ User communication sent
- □ Success metrics defined

□ Stakeholder sign-off

Proof of Concept

ElectricSQL Local-First Sync Demo

---

1. POC Objectives

```
┌──────────────────────────────────────────────────────────────┐
│                    POC
OBJECTIVES                          │
├──────────────────────────────────────────────────────────────┤
│
│
│  1. Validate ElectricSQL with
PostgreSQL                     │
│  2. Demonstrate real-time sync across
devices                │
│  3. Test offline
capability                           │
│  4. Measure sync
performance                          │
│  5. Identify integration
challenges                    │
│
│
│  SUCCESS
CRITERIA:                                   │
│  ✓ Sync 10,000 barcodes in < 60
seconds                   │
│  ✓ Offline writes sync correctly on
reconnect              │
```

```
│  ✓ Conflicts resolved
automatically                         │
│  ✓ No data loss in any
scenario                             │
│
│
└─────────────────────────────────────────────────────────────┘
```

---

2. POC Architecture

```
┌─────────────────────────────────────────────────────────────┐
│                    POC
ARCHITECTURE                          │
├─────────────────────────────────────────────────────────────┤
│
│
│  ┌─────────────┐     ┌─────────────┐
┌─────────────┐   │
│  │   Device A  │     │   Device B  │     │   Device C
│   │
│  │  (Mobile)   │     │  (Tablet)   │     │   (Web)
│   │
│  │  SQLite     │     │  SQLite     │     │  IndexedDB
│   │
│  └──────┬──────┘     └──────┬──────┘
└──────┬──────┘   │
│         │                   │
│           │
│
└───────────────────┼───────────────────┘           │
│
│                               │
│
┌───────┴───────┐                       │
│                     │  Electric
│                       │
│                     │  (Docker)
│                       │
│
└───────┬───────┘                       │
│
│                               │
│
┌───────┴───────┐                       │
│                     │  PostgreSQL
│                       │
│                     │  (Docker)
│                       │
│
└───────────────┘                       │
│
│
└──────────────────────────────────────────────────────────────┘
```

---

3. Setup Instructions

3.1 Docker Compose

```yaml
# docker-compose.poc.yaml
version: '3.8'

services:
  postgres:
    image: postgres:16
    environment:
      POSTGRES_DB: barcode_poc
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    command:
      - postgres
      - -c
      - wal_level=logical
    ports:
      - "5432:5432"
    volumes:
      - pg_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql

  electric:
    image: electricsql/electric:latest
    environment:
      DATABASE_URL: postgresql://postgres:postgres@postgres:5432/barcode_poc
      AUTH_MODE: insecure  # For POC only
      ELECTRIC_WRITE_TO_PG_MODE: direct_writes
    ports:
      - "5133:5133"
    depends_on:
      - postgres

volumes:
  pg_data:
```

3.2 Database Schema

```
-- init.sql
CREATE SCHEMA IF NOT EXISTS app_barcode;

CREATE TABLE app_barcode.bc_batch (
    id SERIAL PRIMARY KEY,
    vendor_code VARCHAR(100) NOT NULL,
    name VARCHAR(300) NOT NULL,
    is_active BOOLEAN DEFAULT FALSE,
    is_deleted BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

```sql
CREATE TABLE app_barcode.bc_parent (
    id SERIAL PRIMARY KEY,
    vendor_code VARCHAR(100) NOT NULL,
    batch_id INTEGER REFERENCES app_barcode.bc_batch(id),
    code VARCHAR(200) NOT NULL,
    qty INTEGER DEFAULT 0,
    is_deleted BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE app_barcode.bc_barcode (
    id SERIAL PRIMARY KEY,
    vendor_code VARCHAR(100) NOT NULL,
    batch_id INTEGER REFERENCES app_barcode.bc_batch(id),
    parent_id INTEGER REFERENCES app_barcode.bc_parent(id),
    barcode VARCHAR(200) UNIQUE NOT NULL,
    note TEXT,
    is_deleted BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Seed data
INSERT INTO app_barcode.bc_batch (vendor_code, name,
        is_active)
VALUES ('VENDOR_A', 'POC Batch 1', true);

INSERT INTO app_barcode.bc_parent (vendor_code, batch_id,
        code, qty)
SELECT 'VENDOR_A', 1, 'PARENT-' || i, 10
FROM generate_series(1, 100) AS i;

INSERT INTO app_barcode.bc_barcode (vendor_code, batch_id,
        barcode)
SELECT 'VENDOR_A', 1, 'BC-' || LPAD(i::text, 6, '0')
FROM generate_series(1, 10000) AS i;
```

3.3 Client Setup (React)

```
# Create POC client
npx create-vite@latest poc-client --template react-ts
cd poc-client

# Install dependencies
npm install @electric-sql/client better-sqlite3
npm install -D @electric-sql/cli
```

3.4 Client Code

```
// src/electric.ts
import { ElectricClient, ShapeStream } from '@electric-sql/client'

const ELECTRIC_URL = 'http://localhost:5133'

export async function createShape(batchId: number) {
  const stream = new ShapeStream({
    url: `${ELECTRIC_URL}/v1/shape`,
    params: {
      table: 'app_barcode.bc_barcode',
      where: `batch_id = ${batchId} AND is_deleted = false`
    }
  })

  return stream
}

// src/App.tsx
import { useState, useEffect } from 'react'
import { createShape } from './electric'

function App() {
  const [barcodes, setBarcodes] = useState([])
  const [syncStatus, setSyncStatus] =
        useState('connecting')
  const [lastSync, setLastSync] = useState(null)

  useEffect(() => {
    const initSync = async () => {
      const shape = await createShape(1)

      shape.subscribe((messages) => {
        // Apply changes to local state
        messages.forEach(msg => {
          if (msg.headers.operation === 'insert') {
            setBarcodes(prev => [...prev, msg.value])
          } else if (msg.headers.operation === 'update') {
            setBarcodes(prev =>
              prev.map(b => b.id === msg.value.id ?
        msg.value : b)
            )
          }
        })

        setSyncStatus('synced')
        setLastSync(new Date())
      })
    }

    initSync()
  }, [])

  return (
    <div>
      <h1>ElectricSQL POC</h1>
      <p>Status: {syncStatus}</p>
      <p>Last sync: {lastSync?.toLocaleTimeString()}</p>
      <p>Barcodes: {barcodes.length}</p>

      <ul>
        {barcodes.slice(0, 20).map(b => (
          <li key={b.id}>{b.barcode}</li>
        ))}
      </ul>
    </div>
  )
}
```

---

4. Test Scenarios

4.1 Initial Sync Test

```
SCENARIO: First device sync
GIVEN: 10,000 barcodes in PostgreSQL
WHEN: Client connects to Electric
THEN: All 10,000 barcodes sync to client
EXPECTED: < 60 seconds
```

4.2 Real-time Sync Test

```
SCENARIO: Insert on one device, see on another
GIVEN: Two devices connected
WHEN: Device A inserts new barcode via PostgreSQL
THEN: Device B sees new barcode within 1 second
```

4.3 Offline Write Test

```
SCENARIO: Write while offline
GIVEN: Device connected and synced
WHEN:
  1. Disconnect network
  2. Update barcode note
  3. Reconnect network
THEN:
  1. Local update succeeds immediately
  2. Update syncs to server on reconnect
```

4.4 Conflict Test

```
SCENARIO: Concurrent edits
GIVEN: Two devices with same barcode
WHEN:
  1. Both devices go offline
  2. Device A: note = "QC OK"
  3. Device B: note = "Shipped"
  4. Both reconnect
THEN:
  1. LWW resolves conflict (latest timestamp wins)
  2. Both devices show same final value
```

---

5. Performance Benchmarks

5.1 Sync Speed

Data Volume	Expected Time	Actual Time	Status
1,000 rows	< 5 sec	TBD	⏳
10,000 rows	< 60 sec	TBD	⏳
100,000 rows	< 5 min	TBD	⏳

5.2 Latency

Operation	Expected	Actual	Status
Local write	< 10ms	TBD	⏳

Operation	Expected	Actual	Status
Sync propagation	< 500ms	TBD	⏳
Reconnection	< 3 sec	TBD	⏳

---

6. Run POC

```
# 1. Start services
docker-compose -f docker-compose.poc.yaml up -d

# 2. Wait for Electric to connect
docker logs -f electric

# 3. Start client
cd poc-client
npm run dev

# 4. Open browser
open http://localhost:5173
```

---

7. POC Findings Template

```
## POC Results

### Date: ____

### Success Criteria

| Criteria | Met | Notes |
|----------|-----|-------|
| Sync 10K in < 60s | □ | |
| Offline writes work | □ | |
| Conflicts resolved | □ | |
| No data loss | □ | |

### Performance Results

Initial sync (10K rows): ____ seconds
Real-time propagation: ____ ms
Reconnection time: ____ seconds

### Issues Found

1. ____
2. ____

### Recommendations

1. ____
2. ____

### Go/No-Go Decision

[ ] GO - Proceed to Phase 1
[ ] NO-GO - Reason: ____
```

P2P / LAN Sync

Device-to-Device Synchronization on Local Network

---

1. Konsep P2P Sync

```
┌──────────────────────────────────────────────────────────────┐
│                     P2P SYNC
CONCEPT                          │
├──────────────────────────────────────────────────────────────┤
│
│
│
SCENARIO:
│
│  • 2 operator di warehouse yang
sama                        │
│  • Terhubung ke WiFi yang
sama                              │
│  • Server/internet tidak stabil atau tidak
ada              │
│  • Butuh sync antar device untuk
pairing                    │
│
│
│  SOLUTION: P2P SYNC via
LAN                                  │
│
│
│  ┌──────────┐      WiFi LAN
┌──────────┐              │
│  │ Device A │◄──────────────────►│ Device B
│              │
│  │ (Scan    │    Direct sync     │ (Scan
│              │
│  │  barcode)│    no server!      │  parent)
│              │
│  └──────────┘
└──────────┘              │
│
│
```

```
│  HYBRID
MODEL:                                                     │
│  1. Primary: Sync via Electric
(internet)                            │
│  2. Fallback: P2P sync via LAN (when
offline)                        │
│  3. Eventually: All sync to server when internet
back       │
│
│
└──────────────────────────────────────────────────────────────┘
```

---

2. Use Cases

2.1 Warehouse Pairing Scenario

```
┌──────────────────────────────────────────────────────────────┐
│             PAIRING USE
CASE                                    │
├──────────────────────────────────────────────────────────────┤
│
│
│
SETUP:
│
│  • Device A: Scan individual
barcodes                         │
│  • Device B: Scan parent
boxes                                │
│  • Both on same WiFi, internet slow/
unavailable                │
│
│
│
WORKFLOW:
```

```
│
│  1. Device B scans Parent-001 (creates parent
record)       │
│  2. P2P sync: Device A receives
Parent-001                  │
│  3. Device A scans BC-001, pairs to
Parent-001              │
│  4. P2P sync: Device B sees BC-001
paired                   │
│  5. Both devices have consistent
view                       │
│  6. When internet back → both sync to
server                │
│
│
│
TIMELINE:
│
│
│
│  Device B    Device A
Server                                   │
│     │           │
│                                     │
│     │──(scan)──►│           │
P2P                         │
│     │◄──────────│           │
sync                        │
│     │
│──(scan)──►│                               │
│     │◄──────────│
│                                  │
│     │           │
│                                     │
│     │═══════════════════════│
Internet                    │
│     │           │           │
```

```
restored                    │
│
├──────────────────────▶│                               │
│     │
├──────────▶│                               │
│     │           │
│                               │
│
│
└─────────────────────────────────────────────────────────────┘
```

2.2 Field Operations

```
SCENARIO: Event/Exhibition venue
- No reliable internet
- 5 operators scanning products
- Need real-time inventory updates

P2P SOLUTION:
- One device acts as "local hub"
- Other devices sync to local hub
- Hub syncs to server when internet available
```

---

3. Technical Architecture

3.1 Discovery Protocol

```
┌─────────────────────────────────────────────────────────────┐
│              DEVICE
DISCOVERY                                │
├─────────────────────────────────────────────────────────────┤
│
│
│  OPTION A: mDNS/Bonjour
```

```
(Recommended)                           │
│
────────────────────────────────────
│
│  • Standard protocol for LAN
discovery                          │
│  • Works on iOS, Android,
Desktop                               │
│  • No configuration
needed                                      │
│
│
│  Device A
broadcasts:                                            │
│    _barcode-
sync._tcp.local                                      │
│    Port:
8765                                                      │
│    TXT: vendor=ABC,
batch=123                                   │
│
│
│  Device B
discovers:                                             │
│    "Found device at
192.168.1.100:8765"                         │
│    Same vendor? Same batch? →
Connect                            │
│
│
│
─────────────────────────────────────────────
│
│
│
│  OPTION B: Broadcast
UDP                                         │
```

```
│
────────────────────────
│
│  • Send discovery packet to
255.255.255.255                 │
│  • Simpler
implementation                                │
│  • May not work on some
networks                            │
│
│
│
─────────────────────────────────────────────
│
│
│
│  OPTION C: QR Code
Pairing                                   │
│
─────────────────────────
│
│  • Device A shows QR with
IP:PORT                               │
│  • Device B scans QR to
connect                               │
│  • Most reliable, no auto-
discovery                             │
│
│
└──────────────────────────────────────────────────────────────┘
```

3.2 Sync Protocol

```
┌──────────────────────────────────────────────────────────────┐
│              P2P SYNC
PROTOCOL                                       │
├──────────────────────────────────────────────────────────────┤
```

```
│
│
│
CONNECTION:
│
│  • WebSocket over
LAN                                       │
│  • TLS with self-signed cert (peer
verification)            │
│  • JWT auth (same token as
server)                          │
│
│
│  SYNC
FLOW:                                                  │
│
│
│  1.
HANDSHAKE                                               │
│     A → B: {type: "hello", vendor: "ABC", batch:
123,       │
│             checkpoint:
"abc123"}                            │
│     B → A: {type: "hello", vendor: "ABC", batch:
123,       │
│             checkpoint:
"def456"}                            │
│
│
│  2. DELTA
EXCHANGE                                           │
│     A → B: {type: "ops", data: [CRDT ops since B's
point]}  │
│     B → A: {type: "ops", data: [CRDT ops since A's
point]}  │
│
│
```

```
│  3. CONTINUOUS
SYNC                                              │
│     A → B: {type: "op", table:
"bc_barcode",                        │
│             operation: "UPDATE", row:
{...}}                     │
│     B → A: {type: "ack", id:
123}                             │
│
│
│  4. CONFLICT
RESOLUTION                                          │
│     Same CRDT rules as server
sync                           │
│     LWW for most
fields                                         │
│     First-write-wins for
pairing                            │
│
│
└──────────────────────────────────────────────────────────────┘
```

3.3 Mesh Network

```
┌──────────────────────────────────────────────────────────────┐
│              MESH
TOPOLOGY                                      │
├──────────────────────────────────────────────────────────────┤
│
│
│  STAR (Simple):                MESH
(Robust):                    │
│
│
│       B     C                    B ───
C                      │
│        \   /                    │\     /
│                      │
│         \ /                    │ \  /
│                      │
│          A (hub)               │  \/
│                      │
│         / \                    │  /\
│                      │
│        /   \                   │ /  \
│                      │
│       D     E                  D ────
E                      │
│
│
│  Hub advantages:              Mesh
advantages:                 │
│  • Simple sync logic          • No single point of
failure  │
│  • Lower bandwidth            • Any device can
leave        │
│  • Easy to manage             • More
resilient              │
│
│
│  RECOMMENDATION: Star for < 5 devices, Mesh for >
5         │
│
│
└─────────────────────────────────────────────────────────────┘
```

4. Implementation Options

4.1 Technology Stack

Component	Option 1	Option 2	Recommendation
Discovery	mDNS	QR Code	mDNS + QR fallback
Transport	WebSocket	WebRTC	WebSocket
Protocol	Custom CRDT	Y.js	Y.js (battle-tested)
Encryption	TLS	DTLS	TLS

4.2 Y.js Based Implementation

```
// p2p-sync.ts
import * as Y from 'yjs'
import { WebsocketProvider } from 'y-websocket'
import { IndexeddbPersistence } from 'y-indexeddb'

// Shared document for sync
const ydoc = new Y.Doc()

// Local persistence
const persistence = new IndexeddbPersistence('barcode-sync', ydoc)

// P2P connection (when peer discovered)
function connectToPeer(peerUrl: string) {
  const wsProvider = new WebsocketProvider(peerUrl,
        'barcode-room', ydoc)

  wsProvider.on('sync', (isSynced: boolean) => {
    if (isSynced) {
      console.log('Synced with peer!')
    }
  })

  return wsProvider
}

// Shared data structures
const barcodes = ydoc.getMap('barcodes')
const parents = ydoc.getMap('parents')
const pairings = ydoc.getMap('pairings')

// Add barcode
function addBarcode(id: string, data: any) {
  barcodes.set(id, data)
}

// Pair barcode to parent (first-write-wins)
function pairBarcode(barcodeId: string, parentId: string) {
  const existing = pairings.get(barcodeId)
  if (!existing) {
    pairings.set(barcodeId, {
      parentId,
      pairedAt: Date.now(),
      pairedBy: getCurrentUserId()
    })
  }
}

// Listen for changes from peers
barcodes.observe(event => {
  event.changes.keys.forEach((change, key) => {
    if (change.action === 'add') {
      console.log(`New barcode from peer: ${key}`)
      updateUI()
    }
  })
})
```

4.3 WebRTC Alternative

```
// For direct browser-to-browser (no local server needed)
import { WebrtcProvider } from 'y-webrtc'

const provider = new WebrtcProvider('barcode-room', ydoc, {
  signaling: ['wss://signaling.barcode-app.com'],  //
        Fallback
  // For LAN-only, use local signaling or ICE candidates
})
```

---

5. Security Considerations

5.1 P2P Security Model

```
┌──────────────────────────────────────────────────────────────┐
│               P2P
SECURITY                                            │
├──────────────────────────────────────────────────────────────┤
│
│
│
AUTHENTICATION:
│
│  • Same JWT token required for
P2P                             │
│  • Validate vendor_code
matches                                  │
│  • Validate batch
assignment                                     │
│
│
│
AUTHORIZATION:
│
```

```
│  • Only sync data for shared
batches                        │
│  • Cannot access other vendor's
data                           │
│
│
│
ENCRYPTION:
│
│  • TLS for all P2P
connections                               │
│  • Certificate pinning between known
devices                   │
│
│
│  TRUST
MODEL:                                                  │
│  • Device must be registered with server
first               │
│  • P2P only allowed for known device
pairs                   │
│  • Server can revoke P2P capability
remotely                 │
│
│
│  RISK
MITIGATION:                                             │
│  • Audit log of all P2P
syncs                                │
│  • Anomaly detection (unusual sync
volume)                  │
│  • Time-limited P2P sessions (auto-
disconnect)              │
│
│
└──────────────────────────────────────────────────────────────┘
```

6. Hybrid Sync Flow

```
┌──────────────────────────────────────────────────────────────┐
│              HYBRID SYNC DECISION
TREE                       │
├──────────────────────────────────────────────────────────────┤
│
│
│
START
│
│
│                                                         │
│
▼                                                         │
│
┌──────────────────┐
│
│  │ Internet
│                                       │
│  │ available?
│                                       │
│
└────────┬─────────┘
│
│
│                                                   │
│     YES   │
NO                                             │
│     ▼     │
▼                                              │
│  ┌────────┴───┐
┌────────────┐                             │
│  │ Sync via   │  │ Peers on
│                              │
│  │ Electric   │  │ same LAN?
│                              │
│  │ (primary)  │
└─────┬──────┘                             │
│  └────────────┘
│                                     │
│                   YES  │
NO                                │
│                   ▼    │
▼                                 │
│            ┌──────┴────┐
┌────────────┐                    │
│            │ P2P sync  │  │ Queue all
│                    │
│            │ via LAN   │  │ operations
│                    │
│            └───────────┘  │ locally
│                    │
│
└────────────┘                    │
│
│
│  PRIORITY
ORDER:                                             │
│  1. Internet → Electric
(authoritative)                     │
│  2. LAN → P2P sync (temporary, merge
later)                 │
│  3. Offline → Local queue (sync when
possible)              │
│
│
│  MERGE
STRATEGY:                                             │
│  When internet
returns:                                      │
│  1. P2P changes treated as "local
changes"                  │
│  2. All merge to server via
```

```
Electric                                 │
│  3. CRDT handles any
conflicts                                         │
│
│
└─────────────────────────────────────────────────────────────┘
```

---

7. UI/UX for P2P

7.1 P2P Status Indicator

```
┌───────────────────────────────────────────┐
│ ≡  Barcode Scanner                        │
├───────────────────────────────────────────┤
│                                           │
│  Sync Status:                             │
│  ┌─────────────────────────────────────┐  │
│  │ 📶 Server: Offline                    │  │
│  │ 🔗 P2P: 2 devices connected           │  │
│  │     • Device-B (Dewi)                │  │
│  │     • Device-C (Budi)                │  │
│  └─────────────────────────────────────┘  │
│                                           │
│  [Disconnect P2P] [Show QR to Connect]    │
│                                           │
└───────────────────────────────────────────┘
```

7.2 Connection Flow

```
Step 1: Discover
┌────────────────────────────┐
│ Looking for nearby devices │
│ 🔍 Scanning...              │
│                            │
│ Found:                     │
│ ○ Device-B (Dewi)          │
│ ○ Device-C (Budi)          │
│                            │
│ [Connect All] [Select]     │
└────────────────────────────┘

Step 2: Confirm
┌────────────────────────────┐
│ Connect to Device-B?       │
│                            │
│ Vendor: ABC                │
│ Batch: POC-001             │
│ User: Dewi                 │
│                            │
│ [Cancel] [Connect]         │
└────────────────────────────┘

Step 3: Connected
┌────────────────────────────┐
│ ✓ Connected to Device-B    │
│                            │
│ Syncing 150 items...       │
│ ████████░░░░░░ 60%         │
│                            │
└────────────────────────────┘
```

---

8. Implementation Phases

Phase	Scope	Duration
Phase 1	QR-based manual connection	Week 1
Phase 2	mDNS auto-discovery	Week 2

Phase	Scope	Duration
Phase 3	Y.js CRDT sync	Week 3
Phase 4	Hybrid Electric + P2P	Week 4
Phase 5	Testing & hardening	Week 5

---

9. Pros & Cons

Advantages

- ✅ Works without internet
- ✅ Lower latency (LAN is faster)
- ✅ Reduces server load
- ✅ Better UX for field operations

Disadvantages

- ❌ More complex implementation
- ❌ Security requires careful design
- ❌ Debugging harder with multiple sync paths
- ❌ Merge conflicts possible between P2P and server

---

10. Recommendation

```
┌──────────────────────────────────────────────────────────────┐
│
RECOMMENDATION                                         │
├──────────────────────────────────────────────────────────────┤
│
│
│  PHASE 1 (MVP): Server-only sync
(ElectricSQL)                           │
```

```
│  • Get basic local-first working
first                      │
│  • Validate CRDT
approach                                   │
│
│
│  PHASE 2 (v1.1): Add P2P as
enhancement                     │
│  • After MVP stable, add P2P
capability                     │
│  • Start with QR-based manual
connection                    │
│  • Use Y.js for proven CRDT
implementation                  │
│
│
│  PHASE 3 (v1.2): Auto-
discovery                             │
│  • mDNS for seamless device
finding                         │
│  • Mesh network for larger
teams                            │
│
│
│  This staged
approach:                                       │
│  • Reduces initial
complexity                               │
│  • Allows learning from production
usage                    │
│  • P2P builds on proven
foundation                          │
│
│
└─────────────────────────────────────────────────────────────┘
```