Phase 1 Implementation Guide: Background Tracking + Offline Orders

Date: February 17, 2026
Status: Foundation Complete (Ready for Integration)
Phase: 1 of 4

☑ What Was Built

Database Schema (`supabase/migrations/001_core_schema.sql`)

- ✅ `journeys` table - Journey lifecycle management
- ✅ `orders` table - Order persistence
- ✅ `offline_queue` table - Data sync buffer
- ✅ `towns_on_route` table - Dynamic routing + order control
- ✅ `location_history` table - GPS tracking history
- ✅ `restaurants` table - Restaurant profiles + dashboard access
- ✅ `riders` table - Delivery personnel management
- ✅ `restaurant_notifications` table - Notification queue
- ✅ RLS policies for security

Offline Queue System (`src/lib/offlineQueue.ts`)

- ✅ IndexedDB wrapper for persistent data
- ✅ Queue management (enqueue, sync, mark processed)
- ✅ Location storage with offline caching
- ✅ Order storage with deduplication support
- ✅ Sync metadata tracking
- ✅ Auto-cleanup of old synced data

React Hooks (Frontend Logic)

`src/hooks/useJourneyState.ts`

- ✅ Journey lifecycle (create, complete, cancel)
- ✅ Auto-restore active journey on app open
- ✅ Location updates with offline fallback
- ✅ Sync queued data when online
- ✅ Auto-sync every 30 seconds if online
- ✅ Queue statistics tracking

`src/hooks/useBackgroundTracking.ts`

- ✅ Capacitor GPS integration
- ✅ Continuous tracking even when app minimized

- ☑ Permissions handling
- ☑ App state monitoring (foreground/background)
- ☑ Error handling and fallback

`src/hooks/useOrderSync.ts`

- ☑ Create orders with offline_id for deduplication
- ☑ Store orders locally in IndexedDB
- ☑ Sync to server with conflict detection
- ☑ Load journey orders (server + offline)
- ☑ Auto-sync when coming online
- ☑ Pending orders counter

Backend Edge Functions

`supabase/functions/update-location.ts`

- ☑ Location update handler
- ☑ Town proximity calculation (Haversine formula)
- ☑ Automatic town status transitions
  - OPEN → CLOSING_SOON (10 min / 3 km away)
  - CLOSING_SOON → LOCKED (bus arrives)
- ☑ Restaurant notifications on town lock
- ☑ Distance/ETA updates for UI

`supabase/functions/send-sms.ts`

- ☑ Africa's Talking SMS integration
- ☑ Restaurant SMS notifications
- ☑ Error handling and logging
- ☑ Webhook security

🚀 Integration Steps

Step 1: Deploy Database Schema

```bash
# Navigate to project
cd c:\Users\zwexm\LPSN\busnstay-journey-map-main

# Run Supabase migrations
npx supabase migration up
# OR manually in Supabase dashboard: SQL Editor → paste 001_core_schema.sql
```

Step 2: Deploy Edge Functions

```
# Deploy update-location function
npx supabase functions deploy update-location

# Deploy send-sms function
npx supabase functions deploy send-sms

# Test functions
curl -X POST https://your-project.supabase.co/functions/v1/update-location \
  -H "Authorization: Bearer YOUR_ANON_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "journey_id": "test-journey-id",
    "latitude": -10.3337,
    "longitude": 57.5012,
    "accuracy": 25
  }'
```

Step 3: Install Capacitor (For Background Tracking)

```
npm install -D @capacitor/cli @capacitor/core @capacitor/geolocation
@capacitor/app

# Initialize Capacitor
npx cap init BusNStay com.busnstay.delivery

# Add platforms
npx cap add ios
npx cap add android
```

Step 4: Use Hooks in Your Components

Example: Start Journey & Begin Tracking

```
import { useJourneyState } from '@/hooks/useJourneyState';
import { useBackgroundTracking } from '@/hooks/useBackgroundTracking';
import { useAuth } from '@/hooks/useAuth'; // Your auth hook

export function JourneyScreen() {
  const { user } = useAuth();
  const {
    journey,
    startJourney,
    updateLocation,
    queueStats,
  } = useJourneyState(user?.id);

  const {
    isTracking,
    startTracking,
    stopTracking,
    lastLocation,
  } = useBackgroundTracking(
    async (location) => {
      // Called whenever location updates
      if (journey) {
        await updateLocation(
          location.latitude,
          location.longitude,
          location.accuracy
        );
      }
    }
  );

  const handleStartJourney = async () => {
    try {
      const newJourney = await startJourney(
        'from_stop_id',
        'to_stop_id',
        'bus_id'
      );

      // Start background tracking
      await startTracking({
        journeyId: newJourney.id,
        updateInterval: 30000, // 30 seconds
        enableHighAccuracy: true,
      });
    } catch (err) {
      console.error('Failed to start journey:', err);
    }
  };

  return (
    <div>
      <h1>Journey Tracking</h1>
      {journey ? (
        <>
          <p>Journey Status: {journey.status}</p>
          <p>Last Location: {lastLocation?.latitude}, {lastLocation?.longitude}
</p>
          <p>Queue Stats: {queueStats.pending} pending, {queueStats.total}
total</p>
          {isTracking && <p>📍 Tracking Active</p>}
        </>
      ) : (
        <button onClick={handleStartJourney}>Start Journey</button>
      )}
    </div>
  );
}
```

Example: Create Order (With Offline Support)

```
import { useOrderSync } from '@/hooks/useOrderSync';

export function OrderForm({ journeyId, restaurantId, passengerId }) {
  const deviceId = localStorage.getItem('device_id') || '';
  const { createOrder, pendingOrdersCount } = useOrderSync(passengerId,
deviceId);

  const handleCreateOrder = async (formData) => {
    try {
      const order = await createOrder({
        journey_id: journeyId,
        restaurant_id: restaurantId,
        passenger_id: passengerId,
        stop_id: 'current_town_id',
        items: formData.items,
        total_amount: formData.total,
        status: 'PENDING',
      });

      console.log('Order created:', order);
      alert(`Order placed! (${pendingOrdersCount} offline orders queued)`);
    } catch (err) {
      console.error('Failed to create order:', err);
    }
  };

  return (
    <form onSubmit={handleCreateOrder}>
      {/* Form fields */}
      {pendingOrdersCount > 0 && (
        <p className="warning">
          ⚠ {pendingOrdersCount} orders waiting to sync...
        </p>
      )}
      <button type="submit">Place Order</button>
    </form>
  );
}
```

🔧 Environment Setup

Add to `.env.local`

```
VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY=your-anon-key-here

# Africa's Talking (for SMS)
VITE_AFRICA_TALKING_USERNAME=your-username
VITE_AFRICA_TALKING_API_KEY=your-api-key

# SMS webhook secret (for security)
SMS_WEBHOOK_SECRET=your-random-webhook-secret
```

Set Supabase Secrets

```
# In Supabase dashboard: Settings → Edge Functions → Secrets

AFRICA_TALKING_API_KEY=your-api-key-here
AFRICA_TALKING_USERNAME=your-username-here
SMS_WEBHOOK_SECRET=your-webhook-secret
```

📝 Testing Checklist

- [ ] **Offline Queue**
    - [ ] Create order while offline
    - [ ] Verify stored in IndexedDB
    - [ ] Go online, verify syncs to server
    - [ ] Check no duplicates
- [ ] **Background Tracking**
    - [ ] Start journey
    - [ ] Minimize app (move to background)
    - [ ] GPS continues updating
    - [ ] Come back to app, location is current
- [ ] **Journey Persistence**
    - [ ] Start journey
    - [ ] Force close app
    - [ ] Reopen app
    - [ ] Journey auto-restores to tracking screen
- [ ] **Town Automation**
    - [ ] Journey nearing town (>10 min away)
    - [ ] Town status: OPEN → CLOSING_SOON
    - [ ] Journey entering town (<500m)

    - [ ] Town status: CLOSING_SOON → LOCKED
    - [ ] New orders blocked for locked town
    - [ ] Existing orders still visible

- [ ] **SMS Notifications**
    - [ ] Configure Africa's Talking credentials
    - [ ] Create order at restaurant
    - [ ] Restaurant receives SMS

🎯 Key Features Implemented

☑ Journey Never Stops

- Journey persists on server with status machine
- GPS continues in background (Capacitor)
- App auto-resumes journey on open

☑ Orders Never Disappear

- Created locally in IndexedDB first
- Synced to server when online
- Survives app close, phone restart, network loss
- Deduplication prevents duplicates on reconnect

☑ Offline Capability (Days)

- All updates queued locally
- Auto-syncs when online
- Batch processing for efficiency
- Data auto-cleans after 7 days

☑ Town Auto-Closing

- ETA calculated from GPS position
- Town transitions: OPEN → CLOSING_SOON → LOCKED
- UI blocks new orders when locked
- Existing orders unaffected

☑ Data Integrity

- Conflict-free syncing
- Sequence numbers prevent race conditions
- Idempotent operations (safe to retry)

🚨 Potential Issues & Solutions

Issue	Cause	Solution
Duplicated orders on sync	Device created order offline + online simultaneously	`offline_id` prevents duplication. Always check existing before inserting.
GPS battery drain	Continuous high-accuracy tracking	Use lower accuracy in production, adjust interval to 60s. Consider geofencing.
Large IndexedDB size	Too much historical data	Auto-cleanup runs on sync. Can adjust retention period.
SMS not sending	Africa's Talking not configured	Verify API key + username. Check SMS balance. Test with curl first.
Journey not auto-restoring	App cache cleared	Migration data stored separately. Will recreate on first sync.

📊 Metrics to Monitor

```
// Queue health
const { total, pending, synced } = queueStats;
console.log(`Queue: ${pending}/${total} pending (${synced} synced)`);

// Sync delay
const lastSync = journey?.last_sync_time;
const syncDelayMs = Date.now() - new Date(lastSync).getTime();
console.log(`Last sync: ${syncDelayMs}ms ago`);

// GPS accuracy
const accuracy = lastLocation?.accuracy;
if (accuracy > 100) console.warn('Low GPS accuracy:', accuracy);

// Town closure rate
const townStatuses = towns.map(t => t.status);
const openCount = [...townStatuses].filter(s => s === 'OPEN').length;
console.log(`Open towns: ${openCount}/${towns.length}`);
```

🔐 Security Checklist

- ✅ RLS policies ensure users see only their data
- ✅ Offline queue never stores sensitive data (only IDs)
- ✅ SMS webhook requires Bearer token
- ✅ Edge functions validate authentication
- ✅ Location data encrypted in transit (HTTPS)
- ✅ Rider location only visible to assigned passenger
- ✅ Restaurant only sees their own orders

📋 Next Steps (Phase 2)

Once Phase 1 is tested and working:

1. **Restaurant Dashboard** (Week 2)
    - Web interface for restaurants
    - Order notifications + SMS
    - "Ready for pickup" confirmation
    - Rider assignment view
2. **Town Order Automation** (Already partially implemented!)
    - Fine-tune closure thresholds
    - Add pre-close warnings
    - Restaurant prep time estimation
3. **Rider System** (Week 3-4)
    - Auto-matching orders to nearby riders
    - Rider notifications (push + SMS)
    - Live location sharing
    - Delivery confirmation

📞 Support

Questions or Issues?

- Check offline queue size: `offlineQueue.getQueueStats(deviceId)`
- Monitor location accuracy: `console.log(lastLocation?.accuracy)`
- Verify town closure: Check `towns_on_route.status` in Supabase dashboard
- SMS delivery: Check Africa's Talking dashboard for failed sends

---

Phase 1 Complete! ☑ Ready to integrate into your app.