WhatsApp Urgent Shift Integration - Database Integration Instructions

Project: ACG StaffLink - Dominion Healthcare Services Ltd
Database: Supabase Project `rzzxxkppkiasuouuglaf`
Integration Type: Replace Twilio SMS with WhatsApp Business Cloud API via n8n
Target: Enable first-come-first-served urgent shift assignment via WhatsApp

Executive Summary

This document provides instructions for integrating two n8n workflows with the existing ACG StaffLink database. The workflows replace the current Twilio SMS-based urgent shift notification system with WhatsApp Business Cloud API, maintaining the same first-response-wins logic while eliminating SMS costs.

Core Problem Being Solved:

- Current system uses expensive Twilio SMS (~£0.15-£0.30 per message)
- Need real-time urgent shift broadcasting to available care workers
- First staff member to confirm gets the shift (race condition handling required)
- Must integrate with existing `shifts`, `staff`, and related tables

Solution:
Two n8n workflows handle the complete flow:

1. **Workflow 1:** Broadcasts urgent shifts to eligible staff via WhatsApp
2. **Workflow 2:** Captures responses and assigns shift to first confirmer

Current Database Structure

Existing Tables (Confirmed from Supabase Dashboard)

The ACG StaffLink database contains **24 tables** including:

Core Tables:

- `shifts` - Main shift management table
- `staff` - Staff/care worker profiles
- `timesheets` - Time tracking and clock-in/out records
- `clients` - Client/care home information
- `admin_workflow` - Administrative workflow tracking

- `disputes` - Dispute management

Related Tables:

- `marketplace_shifts` - Shift marketplace listings
- `cron_job_runs` - Scheduled job execution logs
- `shifts_formatted` - Formatted shift view

Key Database Functions (48 total)

Relevant functions that may need consideration:

- `check_shift_overlap()` - Validates shift scheduling conflicts
- `get_user_profile()` - Retrieves user profile information
- `get_user_agency_id()` - Gets agency association
- `is_agency_staff()` - Validates staff permissions
- `handle_updated_at()` - Timestamp management
- `prevent_financial_modification()` - Financial record protection

Database Schema Requirements

New Tables to Create

1. `urgent_shifts` Table

Purpose: Track urgent shifts that need immediate filling

```
CREATE TABLE IF NOT EXISTS public.urgent_shifts (
  shift_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

  -- Shift Details
  location TEXT NOT NULL,
  room_number TEXT,
  care_centre_name TEXT NOT NULL,
  shift_date DATE NOT NULL,
  shift_time TEXT NOT NULL,
  shift_end_time TEXT,
  hourly_rate DECIMAL(5,2) NOT NULL DEFAULT 14.00,

  -- Assignment Tracking
  assigned_staff_id TEXT, -- WhatsApp phone number of assignee
  status TEXT DEFAULT 'pending' CHECK (status IN ('pending', 'assigned', 'cancelled', 'ex
  assigned_at TIMESTAMP,

  -- Relationships
  original_shift_id UUID REFERENCES public.shifts(id) ON DELETE CASCADE,
  client_id UUID REFERENCES public.clients(id),
  agency_id UUID,
  -- Metadata
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW(),
  expires_at TIMESTAMP,

  -- Additional Info
  notes TEXT,
  requirements TEXT
);

-- Critical Index for Race Condition Protection
CREATE INDEX IF NOT EXISTS idx_pending_urgent_shifts
ON public.urgent_shifts(status, created_at DESC)
WHERE assigned_staff_id IS NULL;

-- Index for Performance
CREATE INDEX IF NOT EXISTS idx_urgent_shifts_date
ON public.urgent_shifts(shift_date);

CREATE INDEX IF NOT EXISTS idx_urgent_shifts_status
ON public.urgent_shifts(status);
```

Why This Structure:

- `assigned_staff_id` stores WhatsApp phone number (e.g., "447830365939")
- `status` enum ensures data integrity
- `original_shift_id` links to main `shifts` table if urgent shift originated from existing shift
- Race condition index (`WHERE assigned_staff_id IS NULL`) ensures only one staff can claim shift

2. `shift_notifications_sent` Table

Purpose: Log all WhatsApp notifications sent (audit trail and debugging)

```
CREATE TABLE IF NOT EXISTS public.shift_notifications_sent (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

  -- Notification Details
  shift_id UUID REFERENCES public.urgent_shifts(shift_id) ON DELETE CASCADE,
  staff_id TEXT, -- Staff identifier (could be user_id or phone)
  phone_number TEXT NOT NULL,

  -- Message Tracking
  message_status TEXT DEFAULT 'sent' CHECK (message_status IN ('sent', 'delivered', 'reac
  whatsapp_message_id TEXT, -- WhatsApp API message ID

  -- Timestamps
  sent_at TIMESTAMP DEFAULT NOW(),
  delivered_at TIMESTAMP,
  read_at TIMESTAMP,

  -- Error Tracking
  error_message TEXT,
  retry_count INTEGER DEFAULT 0,

  -- Metadata
  template_name TEXT DEFAULT 'urgent_shift_available'
);

-- Indexes for Querying
CREATE INDEX IF NOT EXISTS idx_notifications_shift
ON public.shift_notifications_sent(shift_id);

CREATE INDEX IF NOT EXISTS idx_notifications_phone
ON public.shift_notifications_sent(phone_number);

CREATE INDEX IF NOT EXISTS idx_notifications_status
ON public.shift_notifications_sent(message_status, sent_at DESC);
```

Why This Structure:

- Provides complete audit trail for compliance
- Enables debugging (which staff received notification, when)
- Tracks message delivery status from WhatsApp API
- `retry_count` helps identify problematic phone numbers

Database Schema Modifications

Modifications to Existing `staff` Table

Add WhatsApp Integration Fields:

```
-- Add WhatsApp phone number (if not exists)
ALTER TABLE public.staff
ADD COLUMN IF NOT EXISTS phone TEXT;

-- Add WhatsApp opt-in status
ALTER TABLE public.staff
ADD COLUMN IF NOT EXISTS whatsapp_opt_in BOOLEAN DEFAULT false;

-- Add urgent shift preferences
ALTER TABLE public.staff
ADD COLUMN IF NOT EXISTS receive_urgent_shifts BOOLEAN DEFAULT true;

-- Add notification settings
ALTER TABLE public.staff
ADD COLUMN IF NOT EXISTS notification_preferences JSONB DEFAULT '{
  "urgent_shifts": true,
  "shift_reminders": true,
  "shift_updates": true
}'::jsonb;

-- Update timestamp trigger (if not already present)
CREATE TRIGGER set_staff_updated_at
```

```
  BEFORE UPDATE ON public.staff
  FOR EACH ROW
  EXECUTE FUNCTION public.handle_updated_at();
```

Important Data Integrity:

- `phone` must be in international format without spaces: `447830365939`
- `whatsapp_opt_in` ensures GDPR compliance
- `receive_urgent_shifts` allows staff to opt-out of urgent notifications
- `notification_preferences` provides granular control

Modifications to Existing `shifts` Table

Add Urgent Shift Tracking:

```
-- Add urgent shift flag
ALTER TABLE public.shifts
ADD COLUMN IF NOT EXISTS is_urgent BOOLEAN DEFAULT false;

-- Add urgent shift reference
ALTER TABLE public.shifts
ADD COLUMN IF NOT EXISTS urgent_shift_id UUID REFERENCES public.urgent_shifts(shift_id);

-- Index for urgent shift queries
CREATE INDEX IF NOT EXISTS idx_shifts_urgent
ON public.shifts(is_urgent) WHERE is_urgent = true;
```

Why These Changes:

- Links regular shifts to urgent shift broadcasts
- Enables reporting on urgent vs regular shift fill rates
- Allows filtering urgent shifts in admin dashboards

n8n Workflow Integration Points

Workflow 1: WhatsApp Urgent Shift Broadcast

File: `whatsapp-urgent-shift-broadcast.json`

Flow:

```
Manual Trigger → Get Urgent Shift → Get Eligible Staff → Loop Over Staff →
Rate Limit → Send WhatsApp → Log Sent → (Loop continues)
```

Database Interactions:

1. **Get Urgent Shift Node (Supabase)**

- **Table:** `urgent_shifts`
- **Operation:** `getAll`
- **Filter:** Most recent pending shift
- **Query:**

```
status=eq.pending
ORDER BY created_at DESC
LIMIT 1
```

2. **Get Eligible Staff Node (Supabase)**
   - **Table:** `staff`
   - **Operation:** `getAll`
   - **Filters:**

   ```
   status=eq.active
   available=eq.true
   whatsapp_opt_in=eq.true
   receive_urgent_shifts=eq.true
   phone=not.is.null
   ```

   - **Critical:** Only get staff who have WhatsApp enabled and valid phone numbers
3. **Loop Over Staff Node**
   - Iterates through staff list one-by-one
   - Ensures sequential message sending (rate limit compliance)
4. **Send WhatsApp Node**
   - Uses WhatsApp Business Cloud API
   - Sends pre-approved template: `urgent_shift_available`
   - Template parameters:
     - `{{1}}` = `staff.first_name`
     - `{{2}}` = `urgent_shifts.location`
     - `{{3}}` = `urgent_shifts.shift_date`
     - `{{4}}` = `urgent_shifts.shift_time`
     - `{{5}}` = `urgent_shifts.shift_id`
5. **Log Sent Node (Supabase)**
   - **Table:** `shift_notifications_sent`
   - **Operation:** `insert`
   - **Data:**

   ```
   {
     "shift_id": "from urgent_shifts",
     "staff_id": "from staff loop",
     "phone_number": "from staff loop",
   ```

```
  "sent_at": "current timestamp",
  "message_status": "sent"
}
```

Configuration Requirements:

```
// Node: Get Eligible Staff
{
  "operation": "getAll",
  "tableId": "staff",
  "returnAll": true,
  "filters": {
    "conditions": [
      { "keyName": "status", "condition": "eq", "keyValue": "active" },
      { "keyName": "available", "condition": "eq", "keyValue": "true" },
      { "keyName": "whatsapp_opt_in", "condition": "eq", "keyValue": "true" },
      { "keyName": "receive_urgent_shifts", "condition": "eq", "keyValue": "true" },
      { "keyName": "phone", "condition": "isNotNull" }
    ]
  }
}
```

Workflow 2: WhatsApp Response Handler

File: `whatsapp-response-handler.json`

Flow:

```
WhatsApp Webhook → Extract Message → Is Confirm? →
  [TRUE] → Get Pending Shift → Still Available? →
    [TRUE] → Assign Shift → Send Confirmation
    [FALSE] → Send Already Taken
  [FALSE] → (End)
```

Database Interactions:

1. **Get Pending Shift Node (Supabase)**
   - **Table:** `urgent_shifts`
   - **Operation:** `getAll`
   - **Filter:** Latest unassigned shift
   - **Query:**
     ```
     status=eq.pending
     assigned_staff_id=is.null
     ORDER BY created_at DESC
     LIMIT 1
     ```
   - **Critical:** Must check `assigned_staff_id IS NULL` to prevent race conditions

2. **Still Available? Node (IF Condition)**
   - Checks if `assigned_staff_id` is empty/null
   - Ensures shift hasn't been claimed while message was in transit
3. **Assign Shift Node (Supabase) - CRITICAL FOR RACE CONDITIONS**
   - **Table:** `urgent_shifts`
   - **Operation:** `update`
   - **Filter String (CRITICAL):**

     ```
     shift_id=eq.{{ shift_id }}&amp;assigned_staff_id=is.null
     ```

   - **Fields to Update:**

     ```
     {
       "assigned_staff_id": "{{ responder_phone }}",
       "status": "assigned",
       "assigned_at": "{{ current_timestamp }}"
     }
     ```

   - **Why Filter String Matters:** The `&amp;assigned_staff_id=is.null` condition ensures that if two staff respond simultaneously, only the first UPDATE succeeds. The second will fail because `assigned_staff_id` is no longer null.
4. **Send Confirmation Node (WhatsApp)**
   - Sends confirmation message to successful responder
   - Free-form message (within 24-hour customer service window)
   - Includes: location, date, time, hourly rate, instructions
5. **Send Already Taken Node (WhatsApp)**
   - Sends to responders who were too slow
   - Encourages them to stay alert for next shift

Configuration Requirements:

```
// Node: Assign Shift (Race-Safe Update)
{
  "operation": "update",
  "tableId": "urgent_shifts",
  "filterType": "string",
  "filterString": "shift_id=eq.{{ $json.shift_id }}&amp;assigned_staff_id=is.null",
  "fieldsUi": {
    "fieldValues": [
      {
        "fieldName": "assigned_staff_id",
        "fieldValue": "={{ $node['Extract Message'].json.from }}"
      },
      {
        "fieldName": "status",
        "fieldValue": "assigned"
```

```
        },
        {
          "fieldName": "assigned_at",
          "fieldValue": "={{ $now.toISO() }}"
        }
      ]
    }
  }
```

Critical Race Condition Handling

The Problem

When multiple staff respond "YES" simultaneously:

1. Both workflows query database for pending shift
2. Both see shift is available (`assigned_staff_id IS NULL`)
3. Without protection, both could be assigned the same shift

The Solution: Database-Level Atomicity

PostgreSQL UPDATE with WHERE Clause:

```
UPDATE urgent_shifts
SET
  assigned_staff_id = '447830365939',
  status = 'assigned',
  assigned_at = NOW()
WHERE
  shift_id = 'abc-123-def-456'
  AND assigned_staff_id IS NULL;  -- CRITICAL LINE
```

How It Works:

1. First staff response reaches database → UPDATE succeeds (1 row affected)
2. Second staff response reaches database → UPDATE fails (0 rows affected, because `assigned_staff_id` is no longer NULL)
3. n8n workflow checks rows affected:
   - If 1 row: Send confirmation
   - If 0 rows: Shift already taken, send "too slow" message

Implementation in n8n:
The `filterString` parameter ensures the WHERE clause includes the null check:

```
"filterString": "shift_id=eq.{{ $json.shift_id }}&amp;assigned_staff_id=is.null"
```

This translates to the SQL WHERE clause that provides atomicity at the database level.

Integration with Existing Application Code

Frontend Integration Points

1. Create Urgent Shift from Admin Dashboard

When admin creates an urgent shift:

```
// POST /rest/v1/urgent_shifts
const urgentShift = await supabase
  .from('urgent_shifts')
  .insert({
    care_centre_name: 'Divine Care Centre',
    location: 'Room 11',
    shift_date: '2025-11-20',
    shift_time: '08:00-20:00',
    hourly_rate: 14.00,
    status: 'pending',
    original_shift_id: existingShiftId, // if converting existing shift
    client_id: clientId,
    agency_id: agencyId
  })
  .select()
  .single();

// The INSERT triggers n8n Workflow 1 (if using Supabase trigger)
// Or manually trigger n8n webhook with shift_id
```

2. Display Urgent Shift Status

Real-time updates on shift assignment:

```
// Subscribe to urgent_shifts changes
const urgentShiftSubscription = supabase
  .channel('urgent-shifts')
  .on(
    'postgres_changes',
    {
      event: 'UPDATE',
      schema: 'public',
      table: 'urgent_shifts',
      filter: `shift_id=eq.${shiftId}`
    },
    (payload) =&gt; {
      if (payload.new.status === 'assigned') {
        // Show "Shift Assigned to [staff_name]" notification
        showNotification({
          type: 'success',
          message: `Shift assigned to staff member`,
          assignedPhone: payload.new.assigned_staff_id
        });
      }
    }
  )
  .subscribe();
```

3. Link Assigned Staff to Staff Profile

Match `assigned_staff_id` (phone number) back to staff record:

```
// After assignment, get staff details
const { data: assignedStaff } = await supabase
  .from('staff')
  .select('id, first_name, last_name, email')
  .eq('phone', urgentShift.assigned_staff_id)
  .single();

// Update original shift with assigned staff
if (urgentShift.original_shift_id) {
  await supabase
    .from('shifts')
    .update({
      assigned_staff_id: assignedStaff.id,
      status: 'assigned'
    })
    .eq('id', urgentShift.original_shift_id);
}
```

Backend Integration Points

1. Trigger n8n Workflow 1 Automatically

Option A: Supabase Database Trigger

```
-- Create function to call n8n webhook
CREATE OR REPLACE FUNCTION notify_n8n_urgent_shift()
RETURNS TRIGGER AS $$
BEGIN
  PERFORM net.http_post(
    url := 'https://your-n8n-instance.com/webhook/urgent-shift-created',
    headers := '{"Content-Type": "application/json"}'::jsonb,
    body := json_build_object(
      'shift_id', NEW.shift_id,
      'location', NEW.location,
      'shift_date', NEW.shift_date,
      'shift_time', NEW.shift_time,
      'hourly_rate', NEW.hourly_rate
    )::jsonb
  );
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Attach trigger to urgent_shifts INSERT
CREATE TRIGGER on_urgent_shift_created
  AFTER INSERT ON public.urgent_shifts
  FOR EACH ROW
  WHEN (NEW.status = 'pending')
  EXECUTE FUNCTION notify_n8n_urgent_shift();
```

Option B: Application-Level Webhook Call

```
// After creating urgent shift
const response = await fetch('https://your-n8n-instance.com/webhook/urgent-shift-created'
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    shift_id: urgentShift.shift_id,
    location: urgentShift.location,
    shift_date: urgentShift.shift_date,
    shift_time: urgentShift.shift_time,
    hourly_rate: urgentShift.hourly_rate
  })
});
```

2. Expire Stale Urgent Shifts

Create scheduled job to mark old pending shifts as expired:

```
-- Function to expire old urgent shifts
CREATE OR REPLACE FUNCTION expire_old_urgent_shifts()
RETURNS void AS $$
BEGIN
  UPDATE public.urgent_shifts
  SET
    status = 'expired',
    updated_at = NOW()
  WHERE
    status = 'pending'
    AND assigned_staff_id IS NULL
    AND created_at &lt; NOW() - INTERVAL '30 minutes';
END;
$$ LANGUAGE plpgsql;

-- Schedule with pg_cron (if available)
SELECT cron.schedule(
  'expire-urgent-shifts',
  '*/5 * * * *', -- Every 5 minutes
  'SELECT expire_old_urgent_shifts();'
);
```

Data Validation and Integrity

Phone Number Validation

Before inserting/updating staff phone numbers:

```
// Phone validation function
function validateWhatsAppPhone(phone) {
  // Remove all non-numeric characters
  const cleaned = phone.replace(/\D/g, '');

  // Must be 10-15 digits (international format)
  if (cleaned.length &lt; 10 || cleaned.length &gt; 15) {
    throw new Error('Invalid phone number length');
  }

  // UK numbers should start with 44
  if (!cleaned.startsWith('44') &amp;&amp; !cleaned.startsWith('1') &amp;&amp; !cleaned.s
    console.warn('Phone number may not be in international format');
  }

  return cleaned; // Return digits only
}

// Usage in staff update
const validatedPhone = validateWhatsAppPhone(inputPhone);
await supabase
  .from('staff')
  .update({ phone: validatedPhone })
  .eq('id', staffId);
```

Shift Data Validation

Before creating urgent shift:

```
// Validation function
function validateUrgentShift(shiftData) {
  const errors = [];

  // Required fields
  if (!shiftData.care_centre_name) errors.push('Care centre name required');
  if (!shiftData.location) errors.push('Location required');
  if (!shiftData.shift_date) errors.push('Shift date required');
  if (!shiftData.shift_time) errors.push('Shift time required');

  // Date validation
  const shiftDate = new Date(shiftData.shift_date);
  const today = new Date();
  today.setHours(0, 0, 0, 0);

  if (shiftDate &lt; today) {
    errors.push('Shift date cannot be in the past');
  }
  // Hourly rate validation
  if (shiftData.hourly_rate &lt; 10 || shiftData.hourly_rate &gt; 50) {
    errors.push('Hourly rate must be between £10 and £50');
  }

  if (errors.length &gt; 0) {
    throw new Error(`Validation errors: ${errors.join(', ')}`);
  }

  return true;
}
```

Testing Strategy

Phase 1: Database Testing

1. Create Test Data

```
-- Insert test urgent shift
INSERT INTO public.urgent_shifts (
  care_centre_name,
  location,
  shift_date,
  shift_time,
  hourly_rate,
  status
) VALUES (
  'Test Care Centre',
  'Room 99',
  CURRENT_DATE + INTERVAL '1 day',
  '09:00-17:00',
  14.00,
  'pending'
);

-- Insert test staff with WhatsApp enabled
INSERT INTO public.staff (
  first_name,
  last_name,
  phone,
  status,
  available,
  whatsapp_opt_in,
  receive_urgent_shifts
) VALUES (
  'Test',
  'Staff',
  '447700900000', -- UK test number
  'active',
  true,
  true,
  true
);
```

2. Test Race Condition Protection

```
-- Simulate simultaneous updates (run in two separate connections)
-- Connection 1:
UPDATE public.urgent_shifts
SET assigned_staff_id = '447700900001', status = 'assigned', assigned_at = NOW()
WHERE shift_id = 'test-shift-id' AND assigned_staff_id IS NULL;

-- Connection 2 (run immediately after):
UPDATE public.urgent_shifts
SET assigned_staff_id = '447700900002', status = 'assigned', assigned_at = NOW()
WHERE shift_id = 'test-shift-id' AND assigned_staff_id IS NULL;

-- Check result:
SELECT * FROM public.urgent_shifts WHERE shift_id = 'test-shift-id';
-- Should show only ONE assigned_staff_id (first one to execute)
```

Phase 2: Workflow Testing

1. Test Workflow 1 (Broadcast)

- Create test urgent shift in database
- Manually trigger n8n Workflow 1
- Verify:
  - All eligible staff queried correctly
  - Loop iterates through all staff
  - WhatsApp messages sent (check WhatsApp Business API logs)
  - `shift_notifications_sent` table populated

2. Test Workflow 2 (Response Handler)

- Activate Workflow 2 webhook
- Send test WhatsApp message: "YES"
- Verify:
  - Webhook receives message
  - Message extracted correctly
  - Pending shift retrieved
  - Shift assigned (check `urgent_shifts.assigned_staff_id`)
  - Confirmation message sent

3. Test Race Condition

- Create urgent shift

- Trigger Workflow 1 to broadcast to 2+ test numbers
- Have both respond "YES" simultaneously
- Verify:
  - Only first responder gets confirmation
  - Others get "already taken" message
  - Only one `assigned_staff_id` in database

Phase 3: Integration Testing

1. End-to-End Flow

```
Admin creates urgent shift in dashboard
  ↓
Database INSERT triggers n8n Workflow 1
  ↓
WhatsApp messages sent to eligible staff
  ↓
First staff responds "YES"
  ↓
Webhook triggers n8n Workflow 2
  ↓
Shift assigned in database
  ↓
Confirmation sent to winner
  ↓
"Already taken" sent to losers
  ↓
Dashboard updates with assigned staff
  ↓
Original shift updated (if applicable)
```

2. Error Scenarios to Test

- No eligible staff available → Handle gracefully
- Invalid phone number → Log error, continue with others
- WhatsApp API failure → Retry logic
- Database connection failure → Error handling
- Expired shift claimed → Reject gracefully

Monitoring and Maintenance

Database Views for Monitoring

```
-- View: Active urgent shifts
CREATE OR REPLACE VIEW public.urgent_shifts_active AS
SELECT
  us.*,
  COUNT(sn.id) as notifications_sent,
  CASE
    WHEN us.status = 'pending' AND us.created_at &lt; NOW() - INTERVAL '30 minutes'
    THEN 'stale'
    ELSE us.status
  END as effective_status
FROM public.urgent_shifts us
LEFT JOIN public.shift_notifications_sent sn ON sn.shift_id = us.shift_id
WHERE us.status IN ('pending', 'assigned')
GROUP BY us.shift_id
ORDER BY us.created_at DESC;

-- View: Notification success rate
CREATE OR REPLACE VIEW public.notification_stats AS
SELECT
  DATE(sent_at) as date,
  COUNT(*) as total_sent,
  COUNT(*) FILTER (WHERE message_status = 'delivered') as delivered,
  COUNT(*) FILTER (WHERE message_status = 'failed') as failed,
  ROUND(
    COUNT(*) FILTER (WHERE message_status = 'delivered')::numeric /
    COUNT(*)::numeric * 100,
    2
  ) as delivery_rate_percent
FROM public.shift_notifications_sent
GROUP BY DATE(sent_at)
ORDER BY date DESC;

-- View: Response times
CREATE OR REPLACE VIEW public.urgent_shift_response_times AS
SELECT
  us.shift_id,
  us.care_centre_name,
  us.created_at as shift_created,
  us.assigned_at as shift_assigned,
  EXTRACT(EPOCH FROM (us.assigned_at - us.created_at)) / 60 as response_time_minutes
FROM public.urgent_shifts us
WHERE us.status = 'assigned'
ORDER BY us.assigned_at DESC;
```

Scheduled Maintenance Tasks

1. Clean Old Notification Logs (Keep 90 days)

```
CREATE OR REPLACE FUNCTION cleanup_old_notifications()
RETURNS void AS $$
BEGIN
  DELETE FROM public.shift_notifications_sent
  WHERE sent_at &lt; NOW() - INTERVAL '90 days';
END;
$$ LANGUAGE plpgsql;

-- Schedule monthly cleanup
SELECT cron.schedule(
  'cleanup-notifications',
  '0 2 1 * *', -- 2 AM on 1st of each month
  'SELECT cleanup_old_notifications();'
);
```

2. Archive Completed Urgent Shifts (After 30 days)

```
CREATE TABLE IF NOT EXISTS public.urgent_shifts_archive (
  LIKE public.urgent_shifts INCLUDING ALL
);

CREATE OR REPLACE FUNCTION archive_old_urgent_shifts()
RETURNS void AS $$
BEGIN
  -- Copy to archive
  INSERT INTO public.urgent_shifts_archive
  SELECT * FROM public.urgent_shifts
  WHERE status IN ('assigned', 'expired', 'cancelled')
    AND updated_at &lt; NOW() - INTERVAL '30 days';

  -- Delete from main table
  DELETE FROM public.urgent_shifts
  WHERE status IN ('assigned', 'expired', 'cancelled')
    AND updated_at &lt; NOW() - INTERVAL '30 days';
END;
$$ LANGUAGE plpgsql;
```

Security Considerations

Row Level Security (RLS)

Enable RLS on New Tables:

```
-- Enable RLS
ALTER TABLE public.urgent_shifts ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.shift_notifications_sent ENABLE ROW LEVEL SECURITY;

-- Policy: Admins can do everything
CREATE POLICY "Admins full access urgent_shifts"
ON public.urgent_shifts
FOR ALL
TO authenticated
USING (public.is_super_admin() OR public.is_agency_admin());

-- Policy: Staff can view their assigned shifts
```

```
CREATE POLICY "Staff view own assignments"
ON public.urgent_shifts
FOR SELECT
TO authenticated
USING (assigned_staff_id = (SELECT phone FROM public.staff WHERE user_id = auth.uid()));

-- Policy: System can insert (for n8n service role)
CREATE POLICY "Service role insert notifications"
ON public.shift_notifications_sent
FOR INSERT
TO service_role
WITH CHECK (true);
```

WhatsApp Webhook Security

Verify Meta Signature:

Add to n8n Workflow 2 after webhook trigger:

```
// Code Node: Verify Meta Signature
const crypto = require('crypto');

const signature = $node["WhatsApp Webhook"].json.headers['x-hub-signature-256'];
const body = JSON.stringify($node["WhatsApp Webhook"].json.body);
const secret = process.env.WHATSAPP_APP_SECRET;

const expectedSignature = 'sha256=' + crypto
  .createHmac('sha256', secret)
  .update(body)
  .digest('hex');

if (signature !== expectedSignature) {
  throw new Error('Invalid webhook signature - possible security breach');
}

return $input.all();
```

Environment Variables

Required Environment Variables for n8n:

```
# WhatsApp Configuration
WHATSAPP_PHONE_NUMBER_ID=your_phone_number_id
WHATSAPP_APP_SECRET=your_app_secret

# Supabase Configuration
SUPABASE_URL=https://rzzxxkppkiasuouuglaf.supabase.co
SUPABASE_SERVICE_ROLE_KEY=your_service_role_key

# Application Settings
NODE_ENV=production
```

Migration Checklist

Pre-Migration Tasks

- [ ] Backup existing database (full dump)
- [ ] Test new tables in development environment
- [ ] Verify phone number data quality in `staff` table
- [ ] Set up WhatsApp Business Account
- [ ] Get WhatsApp message template approved by Meta
- [ ] Configure n8n instance with credentials
- [ ] Test workflows with dummy data

Migration Execution

- [ ] Run database migration scripts (create tables, add columns)
- [ ] Update existing staff records with WhatsApp opt-in status
- [ ] Import n8n workflows
- [ ] Configure workflow credentials
- [ ] Set up webhook endpoints in Meta Business Suite
- [ ] Update frontend to create urgent shifts
- [ ] Enable monitoring dashboards

Post-Migration Validation

- [ ] Send test urgent shift (end-to-end)
- [ ] Verify race condition handling
- [ ] Check notification logs
- [ ] Monitor for 24 hours
- [ ] Compare costs (Twilio vs WhatsApp)
- [ ] Gather staff feedback

Rollback Plan

If issues arise:

1. Disable n8n workflows
2. Revert to Twilio SMS sending
3. Keep new tables (don't lose data)
4. Investigate and fix issues
5. Re-enable when ready

Cost Analysis

Current Costs (Twilio SMS)

Assumptions:

- 50 urgent shifts per month
- Average 10 staff notified per shift
- UK SMS rate: £0.15 per message

Calculation:

```
50 shifts × 10 staff × £0.15 = £75/month
Annual: £900
```

New Costs (WhatsApp Business Cloud API)

Meta Pricing:

- First 1,000 conversations/month: **FREE**
- After 1,000: ~£0.01-£0.05 per conversation

Calculation:

```
50 shifts × 10 staff = 500 conversations/month
Cost: £0 (within free tier)

If scaled to 100 shifts/month:
100 shifts × 10 staff = 1,000 conversations
Cost: Still £0 (at free tier limit)

At 200 shifts/month:
200 shifts × 10 staff = 2,000 conversations
1,000 free + 1,000 paid × £0.03 = £30/month
Annual: £360 (60% savings vs SMS)
```

Additional Benefits:

- Rich media support (images, location)
- Read receipts
- Higher engagement rates
- Better deliverability

Support and Troubleshooting

Common Issues

Issue 1: Phone Number Format Errors

Symptom: WhatsApp messages not sending
Solution: Ensure phone numbers are:

- Digits only (no spaces, dashes, parentheses)
- International format (e.g., 447830365939, not 07830365939)
- Valid country code

Fix:

```
-- Clean up phone numbers
UPDATE public.staff
SET phone = REGEXP_REPLACE(phone, '[^0-9]', '', 'g')
WHERE phone IS NOT NULL;

-- Validate UK numbers
UPDATE public.staff
SET phone = '44' || SUBSTRING(phone FROM 2)
WHERE phone LIKE '0%' AND LENGTH(phone) = 11;
```

Issue 2: Race Condition Still Occurring

Symptom: Multiple staff assigned to same shift
Diagnosis: Check if filter string includes null check
Solution: Verify n8n Update node has:

```
filterString: "shift_id=eq.{{ $json.shift_id }}&amp;assigned_staff_id=is.null"
```

Issue 3: Webhook Not Triggering

Symptom: Responses not captured
Diagnosis:

- Check Meta webhook configuration
- Verify n8n webhook URL is accessible
- Check webhook subscription includes "messages" field

Solution:

```
# Test webhook manually
curl -X POST https://your-n8n-instance.com/webhook/whatsapp-urgent-shift \
  -H "Content-Type: application/json" \
  -d '{"test": "data"}'
```

Issue 4: Template Not Sending

Symptom: Template send fails
Diagnosis: Template not approved or parameters mismatch
Solution:

- Verify template status in Meta Business Suite
- Ensure parameter count matches template
- Check template name spelling

Logging and Debugging

Enable n8n Execution Logging:

In n8n settings:

```
{
  "executions": {
    "saveDataOnSuccess": "all",
    "saveDataOnError": "all",
    "saveExecutionProgress": true
  }
}
```

Query Failed Executions:

```
-- Failed notifications
SELECT * FROM public.shift_notifications_sent
WHERE message_status = 'failed'
ORDER BY sent_at DESC
LIMIT 50;

-- Shifts with no notifications sent
SELECT us.*
FROM public.urgent_shifts us
LEFT JOIN public.shift_notifications_sent sn ON sn.shift_id = us.shift_id
WHERE us.created_at &gt; NOW() - INTERVAL '7 days'
  AND sn.id IS NULL;

-- Response time analysis
SELECT
  AVG(EXTRACT(EPOCH FROM (assigned_at - created_at)) / 60) as avg_response_minutes,
  MIN(EXTRACT(EPOCH FROM (assigned_at - created_at)) / 60) as fastest_response_minutes,
  MAX(EXTRACT(EPOCH FROM (assigned_at - created_at)) / 60) as slowest_response_minutes
FROM public.urgent_shifts
WHERE status = 'assigned'
  AND assigned_at IS NOT NULL;
```

Future Enhancements

Phase 2 Features

1. Staff Preferences

Allow staff to set preferred:

- Locations
- Shift times
- Minimum hourly rate
- Maximum distance from home

Filter eligible staff based on preferences before broadcasting.

2. Automated Fallback

If no staff claims shift within X minutes:

- Increase hourly rate by 10%
- Re-broadcast to wider pool
- Eventually escalate to agency manager

3. Shift Confirmations

Require staff to confirm:

- 24 hours before shift
- 1 hour before shift
- When arriving on-site (GPS check-in)

4. Performance Metrics

Track staff:

- Response time to urgent shifts
- Acceptance rate
- Cancellation rate
- Shift completion rate

Use metrics for priority ordering (faster responders get notified first).

5. Multi-Agency Support

Allow multiple agencies to:

- Broadcast to their own staff pools
- Share shifts in marketplace
- Set agency-specific rates

Conclusion

This integration replaces expensive SMS with free WhatsApp messaging while maintaining the critical first-come-first-served logic through database-level race condition protection.

Key Success Factors:

1. Proper phone number formatting and validation
2. Race condition handling via SQL WHERE clause with null check
3. WhatsApp template approval from Meta
4. Comprehensive logging for troubleshooting
5. Gradual rollout with thorough testing

Next Steps for AI Agent:

1. Create new tables (`urgent_shifts`, `shift_notifications_sent`)
2. Add columns to existing tables (`staff.phone`, `staff.whatsapp_opt_in`, etc.)
3. Create database views for monitoring
4. Set up RLS policies
5. Create validation functions
6. Set up scheduled maintenance tasks
7. Configure webhook endpoints
8. Update application code to integrate with new tables

References:

- Workflow 1: `whatsapp-urgent-shift-broadcast.json`
- Workflow 2: `whatsapp-response-handler.json`
- Supabase Project: `rzzxxkppkiasuouuglaf`
- Database: 24 existing tables, 48 existing functions

Document Version: 1.0
Last Updated: November 20, 2025
Author: AI Assistant
For: ACG StaffLink - Dominion Healthcare Services Ltd