WhatsApp UIC Generator - Deployment Guide

Comprehensive Guide for the Democratic Republic of Congo Team

A production-grade WhatsApp bot system for generating Unique Identifier Codes (UICs) in the context of DRC health services. Built with FastAPI and designed for deployment with Meta's WhatsApp Cloud API.

Important: This system uses **Twilio for development/testing** and **Meta Cloud API for production**. See the Deployment Environments section for details.

Table of Contents

Overview

What does this system do?

This WhatsApp bot conducts an interactive conversation to collect user information and generate a deterministic, privacy-preserving Unique Identifier Code (UIC). The same person will always receive the same UIC when providing the same information.

Getting Started

See the QUICKSTART.md file for instructions.

Prerequisites

- git
- Python 3.12+
- Twilio account (development only)
- ngrok installed
- Meta Cloud account

NOTE This README.md is prepared for deployment on a Linux server. For deployment on a Windows server (rare, but possible), there is a rough guide in the FR directory, but I cannot confirm its accuracy.

Clone the Repository

```
git clone https://github.com/drjforrest/whatsapp-uic-generator.git
cd whatsapp-uic-generator
```

1. Install Dependencies

```
python3 -m venv .venv
source .venv/bin/activate  # On Windows: .venv\Scripts\activate
pip install -e .
```

2. Generate Security Salt

```
python scripts/generate_salt.py
```

Copy the output.

3. Configure Environment

NOTE: This configuration is for the development environment ONLY. Twillio allows you to set up the WhatsApp bot and have a functional app without having to register with Meta Cloud. It is not intended for production deployment. See the steps below for production environment configuration.

```
cp .env.example .env
# Edit .env with:
# - Generated salt
# - Twilio credentials (from console.twilio.com)
```

4. Initialize Database

```
python scripts/init_db.py
```

5. Start Services

Terminal 1 - FastAPI:

```
./scripts/run_dev.sh
# OR: uvicorn app.main:app --reload
```

Terminal 2 - ngrok:

```
ngrok http 8000
# Copy the HTTPS URL
```

Key Features

- **Privacy-preserving**: Uses SHA-256 hashing with salt to generate anonymous yet deterministic codes
- **DRC-adapted**: Handles French accents, various name spellings, and local health zone data
- **Interactive flow**: Natural conversation flow in French with validation and error handling
- **Production quality**: Structured logging, database persistence, comprehensive error handling
- **Duplicate detection**: Automatically prevents duplicate registrations
- **Optional QR codes**: Generate scannable QR codes for easy UIC access
- **Easy deployment**: Works with Twilio sandbox for testing, Meta Cloud API for production

Conversation Flow

The bot asks 5 questions in French to generate the UIC code.

UIC Formula: LLLFFFYCG (10 characters)

Questions asked (in French):
1. **Quelles sont les 3 premières lettres de votre nom de famille?** → First 3 letters of last name (LLL)
2. **Quelles sont les 3 premières lettres de votre prénom?** → First 3 letters of first name (FFF)

3. **Quel est le dernier chiffre de votre année de naissance?** → Last digit of birth year (Y)
4. **Quel est le code de votre ville de naissance?** → 2-letter city code (C)
5. **Quel est votre code de genre?** → Gender code: 1, 2, 3, or 4 (G)

UIC Format

The UIC format follows: `LLLFFFYCG` (10 characters total)

- **LLL**: First 3 letters of last name code (e.g., MBE from Mbengue)
- **FFF**: First 3 letters of first name code (e.g., IBR from Ibrahima)
- **Y**: Last digit of birth year (e.g., 7 from 1997)
- **C**: 2-letter city code (e.g., DA for Dakar)
- **G**: Gender code - 1 digit
- 1 = Homme (Male)
- 2 = Femme (Female)
- 3 = Trans
- 4 = Autre (Other)

Example: `MBEIBR7DA1`
- MBE: Last name code “Mbengue”
- IBR: First name code “Ibrahima”
- 7: Born in year ending in 7 (e.g., 1997)
- DA: City code “Dakar”
- 1: Gender code “Homme” (Male)

Deployment Environments

This system is designed to work in two distinct environments:

Development/Testing Environment (Twilio)

Purpose: Quick setup for testing, proof-of-concept, and development

How it works:
- Uses Twilio’s WhatsApp Sandbox or Business API
- Twilio acts as intermediary between your server and WhatsApp
- Webhooks point to your server
- Twilio handles message delivery

Advantages:
- Quick setup (15 minutes)
- No Meta Business verification required initially
- Works with ngrok for local testing
- Sandbox mode available immediately
- All features work including QR codes

Limitations:
- Sandbox requires users to “join” first (send a code)
- Production Twilio WhatsApp requires Meta verification anyway
- Additional cost layer (Twilio fees on top of Meta)

When to use:
- Testing the bot locally
- Proof-of-concept demonstrations
- Development and debugging
- Initial deployment before Meta verification completes

Production Environment (Meta Cloud API)

Purpose: Direct integration with WhatsApp for production deployment

How it works:
- Direct integration with Meta’s WhatsApp Cloud API
- No intermediary service
- Your server communicates directly with Meta’s API
- Webhooks come directly from Meta

Advantages:
- No per-message fees (only Meta's charges)
- Direct API access
- Better performance
- Official WhatsApp Business solution
- No sandbox limitations

Requirements:
- Meta Business Account verification
- WhatsApp Business Account
- Domain with SSL certificate
- Publicly accessible webhook endpoint

When to use:
- Production deployment
- After Meta verification is complete
- When scaling beyond testing phase
- For official DRC health system deployment

Choosing Your Path

Recommended Approach:

1. **Start with Twilio** (Week 1–2)
2. Set up Twilio sandbox
3. Test all functionality
4. Train users
5. Verify DHIS–2 integration
6. **Transition to Meta Before Official Deployment** (Week 3+)
7. Complete Meta Business verification
8. Update webhook configuration
9. Migrate to direct Meta API
10. Remove Twilio dependency

Important Notes:
- The codebase currently uses Twilio's SDK for message sending
- For Meta Cloud API production, you'll need to adapt the `webhook.py` file to use Meta's API format
- All other code (UIC generation, database, validation) remains the same
- See the Migration Guide section for specific code changes needed

---

Technical Prerequisites

Before starting deployment, ensure you have:

Recommended Server Hardware

For production use (100–1000 users/day):

- **CPU**: 2 cores minimum (4 cores recommended)
- **RAM**: 2 GB minimum (4 GB recommended)
- **Storage**: 20 GB minimum (SSD preferred)
- **Network**: Stable connection with static IP

Required Software

- **Operating System**: Ubuntu 22.04 LTS (recommended) or similar
- **Python**: Version 3.12 or higher
- **Database**: PostgreSQL 14+ (for production) or SQLite (for testing)
- **Web Server**: Nginx (for reverse proxy)
- **SSL Certificate**: Let's Encrypt (free)

External Accounts and Services

- **Meta Business Account**: For WhatsApp Business API

- **Twilio Account**: For WhatsApp integration (alternative if direct Meta isn't available)
- **Domain Name**: A domain for your server (e.g., `whatsapp.your-organization.cd`)

Required Technical Skills

The person installing this system should have:

- Basic Linux command-line knowledge
- Experience with server software installation
- Understanding of basic networking concepts (DNS, ports, SSL)

Meta Registration (WhatsApp Business API)

Option 1: Using Twilio (Recommended for getting started)

Twilio offers simplified integration with the WhatsApp Business API.

Step 1.1: Create a Twilio Account

1. Go to https://www.twilio.com/try-twilio
2. Click on **"Sign Up"**
3. Fill out the form:
4. First and last name
5. Professional email address
6. Strong password
7. Phone number for verification
8. Verify your email address
9. Verify your phone number (you'll receive an SMS with a code)

Step 1.2: Initial Twilio Account Setup

1. Log into the Twilio Console
2. Complete the welcome questionnaire:
3. Select **"Messaging"** as the main product
4. Select **"With code"** for the development method
5. Select **"Python"** as the language
6. Skip through the introduction interface

Step 1.3: Access the WhatsApp Sandbox

For testing (POC Phase):

1. In the Twilio Console, go to **"Messaging"** → **"Try it out"** → **"Send a WhatsApp message"**
2. You'll see:
3. A sandbox number (e.g., `+1 415 523 8886`)
4. A join code (e.g., `join side-orbit`)
5. On your WhatsApp phone:
6. Save the sandbox number in your contacts
7. Send the join code to this number
8. Wait for the confirmation message

Sandbox Limitation: The sandbox is excellent for testing, but all users must join the sandbox before using the bot. For production, you must upgrade to a verified WhatsApp Business number.

Step 1.4: Get a WhatsApp Business Number (Production)

For production:

1. In the Twilio Console, go to **"Messaging"** → **"WhatsApp senders"**

2. Click on **“Get WhatsApp Enabled”**
3. Choose your option:
4. **Option A**: Use an existing Twilio number
5. **Option B**: Purchase a new Twilio number
6. Follow the Meta verification process:
7. Your company name
8. Your organization website
9. Use case description
10. Business registration documents (if required)
11. Wait for approval (typically 1–3 business days)

Step 1.5: Retrieve Your Twilio Credentials

1. In the Twilio Console, go to **“Account”** → **“Dashboard”**
2. Note these **important** values:
3. **Account SID**: Starts with `AC...` (e.g., `AC1234567890abcdef1234567890abcd`)
4. **Auth Token**: Click “Show” to reveal (e.g., `1234567890abcdef1234567890abcd`)
5. **WhatsApp Number**: Your sandbox or business number (e.g., +14155238886)

IMPORTANT: These credentials are confidential. Never share them publicly and don’t commit them to Git.

Save These Values: Copy these three values from the Twilio Console. You will need them when configuring the `.env` file in the Environment Variable Configuration section.

```
# From your Twilio Dashboard
TWILIO_ACCOUNT_SID="AC..."

# From your Twilio Dashboard
TWILIO_AUTH_TOKEN="..."

# Your sandbox or production number
TWILIO_WHATSAPP_NUMBER="+..."
```

Option 2: Direct Meta API Access (Advanced)

If you want to connect directly to Meta without Twilio:

Step 2.1: Create a Meta Business Account

1. Go to https://business.facebook.com/
2. Click **“Create account”**
3. Provide your organization information
4. Verify your business with the required documents

Step 2.2: Configure the WhatsApp Business Application

1. In Meta Business Manager, create a new application
2. Add the **“WhatsApp”** product
3. Configure your Business profile:
4. Business name
5. Description
6. Category (select “Healthcare” or similar)
7. Profile photo
8. Get your WhatsApp Business phone number

Step 2.3: Obtain API Keys

1. In WhatsApp application settings, retrieve:
2. **WhatsApp Business Account ID**
3. **Phone Number ID**
4. **Access Token** (permanent)
5. Configure webhook (we’ll do this later)

Save These Values: If using direct Meta API, copy these values. You will need them when configuring the `.env` file.

```
# Your Access Token
META_WHATSAPP_TOKEN="..."
```

```
# Your Phone Number ID
META_PHONE_NUMBER_ID="..."

# Create a secure random string for webhook verification
META_VERIFY_TOKEN="..."
```

Recommendation: To get started, we recommend Option 1 (Twilio) as it's simpler to configure and manage.

DHIS–2 Integration

Important: DHIS–2 Tracker Endpoints

CRITICAL FOR DEPLOYMENT: The application code contains placeholder endpoint URLs that **MUST be replaced** with your actual DHIS–2 Tracker instance endpoints before production use.

Required DHIS–2 Endpoints

Your DHIS–2 Tracker implementation must expose the following endpoints. Replace the placeholder values in the codebase with your actual DHIS–2 host and endpoints:

1. UIC Generation Endpoint

- **Purpose**: Generate a new UIC in DHIS–2 Tracker
- **Placeholder**: `/uic/generate`
- **Your endpoint**: `https://your-dhis2-host.org/api/uic/generate`
- **Method**: POST
- **Required**: Yes

2. UIC Validation Endpoint

- **Purpose**: Check if a UIC already exists before creating
- **Placeholder**: `/uic/validate`
- **Your endpoint**: `https://your-dhis2-host.org/api/uic/validate`
- **Method**: GET or POST
- **Required**: Yes (for duplicate prevention)

3. Client Search Endpoint

- **Purpose**: Search for existing clients in DHIS–2 by attributes
- **Placeholder**: `/clients/search`
- **Your endpoint**: `https://your-dhis2-host.org/api/clients/search`
- **Method**: GET or POST
- **Required**: Yes (for duplicate checking)

4. Client Creation Endpoint

- **Purpose**: Create a new client record in DHIS–2 Tracker
- **Placeholder**: `/clients/create`
- **Your endpoint**: `https://your-dhis2-host.org/api/clients/create`
- **Method**: POST
- **Required**: Yes

5. Duplicates Merge Endpoint

- **Purpose**: Merge duplicate client records when identified
- **Placeholder**: `/duplicates/merge`
- **Your endpoint**: `https://your-dhis2-host.org/api/duplicates/merge`
- **Method**: POST
- **Required**: Optional (but recommended for data quality)

Configuration Steps

1. **Identify Your DHIS–2 Host**
2. Determine your DHIS–2 Tracker base URL
3. Example: `https://dhis2.health.gov.cd` or `https://tracker.ministry-health.cd`
4. **Verify Endpoint Availability**
5. Confirm all required endpoints are available on your DHIS–2 instance
6. Test each endpoint with your DHIS–2 administrator
7. Document authentication requirements (API keys, OAuth tokens, etc.)
8. **Update Configuration Files**
9. Locate the configuration file containing placeholder endpoints
10. Replace each placeholder with your actual endpoint URL
11. Format: `https://your-dhis2-host.org/api/[endpoint-path]`
12. **Configure Authentication**
13. Add DHIS–2 authentication credentials to your environment variables:

```
DHIS2_BASE_URL="https://your-dhis2-host.org"
DHIS2_API_USERNAME="[YOUR_USERNAME]"
DHIS2_API_PASSWORD="[YOUR_PASSWORD]"
# OR for token-based auth:
DHIS2_API_TOKEN="[YOUR_API_TOKEN]"
```

Integration Workflow

The WhatsApp bot integrates with DHIS–2 as follows:

1. **User Completes Questions** → Bot collects UIC components
2. **Validation Check** → Calls `/uic/validate` to check for existing UIC
3. **Client Search** → If needed, calls `/clients/search` to find duplicates
4. **UIC Generation** → Calls `/uic/generate` to create UIC in DHIS–2
5. **Client Creation** → Calls `/clients/create` to register client
6. **Response to User** → Bot delivers UIC to WhatsApp user

Testing DHIS–2 Integration

Before going live, test the integration:

Test UIC validation endpoint:

```
curl -X POST https://your-dhis2-host.org/api/uic/validate -H "Authorization: Bearer YOUR_TOKEN" -H "Content-Type: application/json" -d '{'
```

Test client search:

```
curl -X GET https://your-dhis2-host.org/api/clients/search?uic=KABJEA512M \
  -H "Authorization: Bearer YOUR_TOKEN"

# Test UIC generation
curl -X POST https://your-dhis2-host.org/api/uic/generate \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "last_name_code": "KAB",
    "first_name_code": "JEA",
    "birth_year_digit": "5",
    "city_code": "12",
    "gender_code": "M"
  }'
```

IMPORTANT NOTES

DO NOT deploy to production until all DHIS–2 endpoints are:
1. Properly configured with your actual URLs
2. Tested and verified to be working
3. Secured with appropriate authentication
4. Monitored for performance and availability

Documentation Required: Your DHIS–2 administrator should provide:
- Complete endpoint documentation
- Authentication mechanism details

- Expected request/response formats
- Rate limiting information
- Error handling procedures

Server Configuration

Step 1: Choose Your Hosting

You have several options for hosting this system:

Option A: Cloud Server (Recommended)

Recommended Providers:
- **DigitalOcean**: Simple and economical ($12–24/month)
- **AWS EC2**: Flexible and scalable
- **Google Cloud Platform**: Good integration tools
- **Heroku**: Simplest but more expensive

For DigitalOcean (recommended for beginners):

1. Create an account at digitalocean.com
2. Create a “Droplet” (virtual server):
3. **Image**: Ubuntu 22.04 LTS
4. **Plan**: Basic (2 GB RAM / 2 vCPUs) - $18/month
5. **Datacenter region**: Choose closest (e.g., Amsterdam or Frankfurt for DRC)
6. **Authentication**: SSH Keys (more secure) or Password
7. Wait for the droplet to be created (1–2 minutes)
8. Note your server’s public IP address

Option B: On-Premises/Local Server

If you have your own infrastructure:

1. Install Ubuntu Server 22.04 LTS on your machine
2. Ensure the server has:
3. A stable Internet connection
4. A static public IP address or dynamic DNS service
5. Ports 80 and 443 open in your firewall
6. Configure router to forward ports 80 and 443 to your server

Step 2: SSH Connection to Server

From your local computer:

```
# Replace YOUR_SERVER_IP with your server's actual IP address
ssh root@YOUR_SERVER_IP

# If using an SSH key
ssh -i /path/to/your/key.pem root@YOUR_SERVER_IP
```

Note Your Server Details: After creating your server, save:
- Server IP address (e.g., `167.99.123.45`)
- SSH username (typically `root` or `ubuntu`)
- SSH key location if using key-based authentication

Step 3: System Update

Once connected to the server:

```
# Update package list
sudo apt update

# Upgrade installed packages
sudo apt upgrade -y
```

```
# Reboot if necessary
sudo reboot
```

Reconnect after reboot.

Step 4: Install Python 3.12+

```
# Check installed Python version
python3 --version

# If version is less than 3.12, install from deadsnakes PPA
sudo apt install software-properties-common -y
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt update
sudo apt install python3.12 python3.12-venv python3.12-dev -y

# Verify installation
python3.12 --version
```

Step 5: Install Additional Dependencies

```
# Install pip for Python 3.12
sudo apt install python3.12-distutils -y
curl -sS https://bootstrap.pypa.io/get-pip.py | python3.12

# Install Git
sudo apt install git -y

# Install Nginx (web server)
sudo apt install nginx -y

# Install PostgreSQL (recommended for production)
sudo apt install postgresql postgresql-contrib -y

# Install certbot (for SSL certificates)
sudo apt install certbot python3-certbot-nginx -y
```

Application Installation

Step 1: Clone the Repository

```
# Navigate to home directory
cd /home

# Clone the repository (replace with your actual repository URL)
git clone https://github.com/your-organization/whatsapp-uic-generator.git

# Navigate into the project directory
cd whatsapp-uic-generator
```

Note: Replace `https://github.com/your-organization/whatsapp-uic-generator.git` with the actual URL of your repository.

Step 2: Create Python Virtual Environment

```
# Create virtual environment
python3.12 -m venv .venv

# Activate virtual environment
source .venv/bin/activate

# You should now see (.venv) in your terminal prompt
```

Step 3: Install Python Dependencies

```
# Upgrade pip
pip install --upgrade pip

# Install project dependencies
pip install -r requirements.txt
```

```
# Verify installation
pip list
```

Step 4: Generate Security Salt

```
# Run the salt generation script
python scripts/generate_salt.py

# This will create a cryptographic salt for UIC hashing
# Copy the generated salt - you'll need it for the .env file
```

Environment Variable Configuration

Step 1: Create Environment File

```
# Copy the example environment file
cp .env.example .env

# Edit the environment file
nano .env
```

Step 2: Fill in Configuration Values

Edit the `.env` file with your actual values:

```
# === APPLICATION SETTINGS ===
APP_NAME="WhatsApp UIC Generator"
APP_VERSION="1.0.0"
DEBUG=false
LOG_LEVEL="INFO"

# === SERVER CONFIGURATION ===
HOST="0.0.0.0"
PORT=8000
WORKERS=4

# === DATABASE ===
# For production, use PostgreSQL:
DATABASE_URL="postgresql+asyncpg://username:password@localhost/uic_db"
# For testing, SQLite is fine:
# DATABASE_URL="sqlite+aiosqlite:///./uic_database.db"

# === TWILIO CREDENTIALS ===
# Get these from: https://console.twilio.com/
TWILIO_ACCOUNT_SID="AC1234567890abcdef1234567890abcd"
TWILIO_AUTH_TOKEN="your_auth_token_here"
TWILIO_WHATSAPP_NUMBER="+14155238886"

# === UIC GENERATION ===
# Paste the salt generated by scripts/generate_salt.py
UIC_SALT="your_generated_salt_here_64_characters_long"

# === SESSION MANAGEMENT ===
SESSION_TIMEOUT_MINUTES=30

# === SECURITY ===
ALLOWED_ORIGINS="https://your-domain.com,https://www.your-domain.com"

# === DHIS-2 INTEGRATION ===
DHIS2_BASE_URL="https://your-dhis2-host.org"
DHIS2_API_USERNAME="your_username"
DHIS2_API_PASSWORD="your_password"
# OR use token-based auth:
# DHIS2_API_TOKEN="your_api_token"
```

Configuration Guide: Replace the placeholder values with your actual credentials:
- **TWILIO Credentials:** Use values saved from Step 1.5 in Meta Registration section
- **UIC_SALT:** Use the salt generated by `scripts/generate_salt.py` (just completed in Step 4 above)

- **DHIS2 Credentials:** Get these from your DHIS–2 administrator
- **Domain:** Use your actual domain name (you'll set this up in the Production Deployment section)

Step 3: Set Proper Permissions

```
# Protect the .env file
chmod 600 .env

# Verify permissions
ls -la .env
# Should show: -rw------- (readable/writable only by owner)
```

Production Deployment

Step 1: Database Setup (PostgreSQL)

```
# Switch to postgres user
sudo -u postgres psql

# Create database and user
CREATE DATABASE uic_db;
CREATE USER uic_user WITH PASSWORD 'secure_password_here';
GRANT ALL PRIVILEGES ON DATABASE uic_db TO uic_user;

# Exit psql
\q
```

Update your `.env` file:

```
DATABASE_URL="postgresql+asyncpg://uic_user:secure_password_here@localhost/uic_db"
```

Step 2: Initialize Database

```
# Activate virtual environment if not already activated
source .venv/bin/activate

# Run database migrations
alembic upgrade head

# Verify tables were created
python scripts/init_db.py
```

Step 3: Configure Domain and SSL

Set up Domain DNS

1. Get your server's public IP address:

```
curl ifconfig.me
```

1. In your domain registrar (e.g., Namecheap, GoDaddy):
2. Create an A record pointing to your server IP
3. Example: `whatsapp.your-org.cd` → `YOUR_SERVER_IP`
4. Wait for DNS propagation (5 minutes to 48 hours)

Obtain SSL Certificate

```
# Request certificate from Let's Encrypt
sudo certbot --nginx -d whatsapp.your-org.cd

# Follow the prompts:
# - Enter your email
# - Agree to Terms of Service
# - Choose whether to redirect HTTP to HTTPS (recommended: Yes)

# Verify certificate auto-renewal
sudo certbot renew --dry-run
```

Note: Replace `whatsapp.your-org.cd` with your actual domain name throughout the Nginx and SSL configuration steps.

Step 4: Configure Nginx

Create Nginx configuration:

```
sudo nano /etc/nginx/sites-available/whatsapp-uic
```

Add this configuration:

```
server {
    listen 80;
    server_name whatsapp.your-org.cd;

    # Redirect HTTP to HTTPS
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl;
    server_name whatsapp.your-org.cd;

    # SSL certificates (certbot will configure these)
    ssl_certificate /etc/letsencrypt/live/whatsapp.your-org.cd/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/whatsapp.your-org.cd/privkey.pem;

    # Security headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;

    # Proxy to FastAPI application
    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }
}
```

Enable the site:

```
# Create symbolic link
sudo ln -s /etc/nginx/sites-available/whatsapp-uic /etc/nginx/sites-enabled/

# Test configuration
sudo nginx -t

# Reload Nginx
sudo systemctl reload nginx
```

Step 5: Create Systemd Service

Create a service file for automatic startup:

```
sudo nano /etc/systemd/system/whatsapp-uic.service
```

Add this content:

```
[Unit]
Description=WhatsApp UIC Generator
After=network.target postgresql.service

[Service]
Type=notify
User=root
WorkingDirectory=/home/whatsapp-uic-generator
Environment="PATH=/home/whatsapp-uic-generator/.venv/bin"
ExecStart=/home/whatsapp-uic-generator/.venv/bin/uvicorn app.main:app --host 0.0.0.0 --port 8000 --workers 4
```

```
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Enable and start the service:

```
# Reload systemd
sudo systemctl daemon-reload

# Enable service (start on boot)
sudo systemctl enable whatsapp-uic

# Start service
sudo systemctl start whatsapp-uic

# Check status
sudo systemctl status whatsapp-uic
```

Step 6: Configure Twilio Webhook

1. Go to Twilio Console
2. Navigate to **Messaging** → **Settings** → **WhatsApp Sandbox Settings**
3. In "When a message comes in" field, enter:

```
https://whatsapp.your-org.cd/whatsapp/webhook
```

4. Set HTTP method to **POST**
5. Click **Save**

For production WhatsApp Business number:
1. Go to **Messaging** → **WhatsApp senders**
2. Select your WhatsApp number
3. Configure webhook URL: `https://whatsapp.your-org.cd/whatsapp/webhook`

QR Code Feature (Optional)

Overview

The bot can optionally generate and send QR codes containing the user's UIC via WhatsApp. This makes it easier for users to save and share their codes.

Status: Optional feature, disabled by default

How It Works

When enabled:
1. User completes the 5 questions
2. Bot generates the UIC code (e.g., `MBEIBR7DA1`)
3. Bot creates a QR code image containing the UIC
4. Bot sends both the text UIC and the QR code image
5. User can scan the QR code to quickly access their UIC

Enabling the Feature

Step 1: Install QR Code Library

```
# Activate virtual environment
source .venv/bin/activate

# Install qrcode with PIL support
pip install qrcode[pil]
```

Step 2: Enable in Environment Configuration

Add to your `.env` file:

```
# Enable QR code generation
ENABLE_QR_CODE=true
```

Step 3: Restart Application

```
# If using systemd service
sudo systemctl restart whatsapp-uic

# Or if running manually
# Stop the application and restart it
```

How QR Codes Are Delivered

With Twilio (Development/Testing)

Works immediately - No additional configuration needed

Twilio automatically:
1. Fetches the QR code image from your server
2. Delivers it to the user via WhatsApp

Requirements:
- Your server must be publicly accessible via HTTPS
- QR codes are served at: `https://your-server.com/static/qr_codes/{UIC}.png`

With Meta Cloud API (Production)

Requires adaptation - Additional code changes needed

Two options for Meta Cloud API:

Option 1: Upload to Meta First (Recommended)

```
# 1. Upload QR code to Meta
POST https://graph.facebook.com/v18.0/{phone_number_id}/media

# 2. Get media_id from response
# 3. Send message with media_id
POST https://graph.facebook.com/v18.0/{phone_number_id}/messages
```

Option 2: Use Public URL

```
{
  "messaging_product": "whatsapp",
  "to": "{recipient}",
  "type": "image",
  "image": {
    "link": "https://your-server.com/static/qr_codes/MBEIBR7DA1.png"
  }
}
```

See `QR_CODE_FEATURE.md` for detailed implementation guide for Meta Cloud API.

Storage and Cleanup

Where QR Codes Are Stored

```
# QR codes are saved to:
static/qr_codes/MBEIBR7DA1.png

# Directory structure:
whatsapp-uic-generator/
  static/
    qr_codes/
      MBEIBR7DA1.png
      MOBMAR3KI2.png
    ...
```

Automatic Cleanup (Optional)

QR codes are stored permanently by default. To enable automatic cleanup, add a cron job:

```
# Edit crontab
crontab -e

# Add this line to delete QR codes older than 7 days (runs daily at 2 AM)
0 2 * * * cd /home/whatsapp-uic-generator && .venv/bin/python -c "from app.services.qr_service import QRCodeService; QRCodeService().clear
```

Testing the QR Code Feature

```
# Test QR code generation
python tests/test_qr_service.py

# Expected output:
# QR code generated successfully!
# QR code file verified
# All QR code tests passed!
```

Security Considerations

Public URLs: QR code images are publicly accessible at predictable URLs
- URLs follow pattern: `/static/qr_codes/{UIC}.png`
- Anyone with the URL can view the QR code
- **Mitigation**: UICs are already privacy-preserving (no PII)

Storage: Consider implementing:
- Automatic cleanup after X days
- Cloud storage (S3, Cloudflare R2) for production
- Temporary signed URLs for added security

Troubleshooting

QR codes not appearing?

1. Check feature is enabled:

```
grep ENABLE_QR_CODE .env
# Should show: ENABLE_QR_CODE=true
```

1. Verify dependency installed:

```
pip list | grep qrcode
# Should show: qrcode 7.4.0 or higher
```

1. Check static directory exists:

```
ls -la static/qr_codes/
# Should exist with .gitkeep file
```

1. Check application logs:

```
tail -f logs/app.log | grep "QR code"
# Should show QR generation messages
```

For complete QR code documentation, see `QR_CODE_FEATURE.md` in the repository.

Testing and Validation

Step 1: Test the Health Endpoint

```
# Test if the application is running
curl https://whatsapp.your-org.cd/whatsapp/health

# Expected response:
# {
#   "status": "healthy",
#   "service": "whatsapp-uic-generator",
#   "version": "1.0.0"
# }
```

Step 2: Test WhatsApp Integration

Using Twilio Sandbox

1. Open WhatsApp on your phone
2. Send the join code to the sandbox number (e.g., `join side-orbit` to `+1 415 523 8886`)
3. Wait for confirmation message
4. Send any message to start the UIC generation flow
5. Complete the conversation (example below)
6. Verify you receive a UIC at the end

Using Production Number

1. Simply message your WhatsApp Business number
2. The bot should automatically respond with the welcome message
3. Complete the 5-question flow
4. Verify UIC generation

Example Conversation Flow

The bot asks questions in **French**:

```
Bot: Bienvenue au Générateur CIU!

Je vais vous poser 5 questions pour générer votre Code d'Identification Unique (CIU).

Votre CIU est:
• Unique pour vous
• Privé et sécurisé
• Peut être régénéré si nécessaire

Tapez RESTART pour recommencer.
Tapez HELP pour de l'aide.

Commençons!

Question 1 sur 5:

Quelles sont les 3 premières lettres de votre nom de famille?

Exemple: MBE

User: MBE

Bot: Compris!

Question 2 sur 5:

Quelles sont les 3 premières lettres de votre prénom?

Exemple: IBR

User: IBR

Bot: Compris!

Question 3 sur 5:

Quel est le dernier chiffre de votre année de naissance?

Exemple: 7 (pour 1997)

User: 7

Bot: Compris!

Question 4 sur 5:

Quel est le code de votre ville de naissance?
(2 lettres)
```

```
Exemple: DA (pour Dakar)

User: DA

Bot: Compris!

Question 5 sur 5:

Quel est votre code de genre?

1 = Homme
2 = Femme
3 = Trans
4 = Autre

User: 1

Bot: Merci! J'ai toutes les informations.

Génération de votre CIU sécurisé...
Veuillez patienter...

Bot: Votre Code d'Identification Unique a été généré!

  Votre CIU:
==============
  MBEIBR7DA1
==============

Ce code est maintenant enregistré à votre nom.

Sauvegardez ce code! Vous pouvez le redemander en commençant une nouvelle conversation.

Vous recevrez également un code QR pour un accès facile. [if QR enabled]

Tapez RESTART pour générer un nouveau CIU ou mettre à jour vos informations.
```

Expected UIC: `MBEIBR7DA1`
- MBE = Mbengue (last name)
- IBR = Ibrahima (first name)
- 7 = 1997 (birth year)
- DA = Dakar (city)
- 1 = Homme (gender)

Step 3: Test Database Integration

```
# Activate virtual environment
source .venv/bin/activate

# Check database connection
python -c "from app.database import engine; print('Database connected!')"

# View generated UICs
python scripts/view_uics.py

# Test duplicate detection
# Generate a UIC, then try again with same information
# You should receive the SAME UIC both times
```

Step 4: Load Testing (Optional)

For production readiness, test with simulated load:

```
# Install load testing tool
pip install locust

# Run load test
locust -f tests/load_test.py --host=https://whatsapp.your-org.cd

# Open browser to http://localhost:8089
# Configure number of users and spawn rate
```

Maintenance and Monitoring

Daily Monitoring

Check Application Status

```
# Check service status
sudo systemctl status whatsapp-uic

# View recent logs
sudo journalctl -u whatsapp-uic -n 100 --no-pager

# Follow logs in real-time
sudo journalctl -u whatsapp-uic -f
```

Check Application Logs

```
# View application logs
tail -f /home/whatsapp-uic-generator/logs/app.log

# Search for errors
grep "ERROR" /home/whatsapp-uic-generator/logs/app.log

# View today's activity
grep "$(date +%Y-%m-%d)" /home/whatsapp-uic-generator/logs/app.log
```

Weekly Maintenance

Database Cleanup

```
# Activate virtual environment
source .venv/bin/activate

# Clean up expired sessions (older than 30 minutes)
curl -X POST https://whatsapp.your-org.cd/whatsapp/cleanup

# Backup database
sudo -u postgres pg_dump uic_db > backup_$(date +%Y%m%d).sql
```

Update System Packages

```
sudo apt update
sudo apt upgrade -y
sudo systemctl restart whatsapp-uic
```

Monthly Maintenance

Review Logs and Analytics

```
# Count total UICs generated this month
python scripts/monthly_report.py

# Check for unusual activity
grep "duplicate_detected" /home/whatsapp-uic-generator/logs/app.log | wc -l
```

SSL Certificate Renewal

```
# Certificates auto-renew, but verify:
sudo certbot renew --dry-run

# If renewal fails, manually renew:
sudo certbot renew
sudo systemctl reload nginx
```

Monitoring Dashboard (Optional)

Set up monitoring with Prometheus and Grafana:

```
# Install Prometheus
# Install Grafana
# Configure metrics collection
# Create dashboards for:
# - Total UICs generated
# - Response times
# - Error rates
# - Active conversations
```

Migrating from Twilio to Meta Cloud API

When you’re ready to move from Twilio (development) to Meta Cloud API (production), follow this guide.

Prerequisites

Before migrating:
- Meta Business Account verified
- WhatsApp Business Account created
- Phone number registered with Meta
- Webhook verified with Meta
- Access token obtained

Step 1: Update Environment Variables

Add Meta Cloud API credentials to `.env`:

```
# Meta Cloud API Configuration
META_ACCESS_TOKEN="your_meta_access_token_here"
META_PHONE_NUMBER_ID="your_phone_number_id_here"
META_WEBHOOK_VERIFY_TOKEN="your_chosen_verify_token_here"

# Keep Twilio for fallback (optional)
TWILIO_ACCOUNT_SID="your_account_sid"
TWILIO_AUTH_TOKEN="your_auth_token"
```

Step 2: Update webhook.py

The current `app/api/webhook.py` uses Twilio’s SDK. For Meta Cloud API, you’ll need to:

1. **Replace message sending logic** from Twilio to Meta’s API
2. **Update webhook verification** for Meta’s challenge verification

Current Twilio code:

```
from twilio.twiml.messaging_response import MessagingResponse

twiml_response = MessagingResponse()
twiml_response.message(response_text)
return Response(content=str(twiml_response), media_type="application/xml")
```

New Meta Cloud API code:

```
import httpx
from fastapi import Response

# Send message via Meta Cloud API
async with httpx.AsyncClient() as client:
    response = await client.post(
        f"https://graph.facebook.com/v18.0/{META_PHONE_NUMBER_ID}/messages",
        headers={
            "Authorization": f"Bearer {META_ACCESS_TOKEN}",
            "Content-Type": "application/json"
        },
        json={
            "messaging_product": "whatsapp",
            "to": phone_number,  # Without 'whatsapp:' prefix
            "type": "text",
            "text": {"body": response_text}
        }
    )

return Response(content='{"status": "ok"}', media_type="application/json")
```

Step 3: Add Webhook Verification Endpoint

Meta requires a verification endpoint. Add to `webhook.py`:

```
@router.get("/webhook")
async def verify_webhook(
    hub_mode: str = Query(None, alias="hub.mode"),
    hub_verify_token: str = Query(None, alias="hub.verify_token"),
    hub_challenge: str = Query(None, alias="hub.challenge")
):
    """Verify webhook for Meta Cloud API."""
    if hub_mode == "subscribe" and hub_verify_token == META_WEBHOOK_VERIFY_TOKEN:
        return Response(content=hub_challenge, media_type="text/plain")
    raise HTTPException(status_code=403, detail="Verification failed")
```

Step 4: Update Message Parsing

Meta's webhook payload is different from Twilio's:

Meta webhook structure:

```
{
  "object": "whatsapp_business_account",
  "entry": [{
    "changes": [{
      "value": {
        "messages": [{
          "from": "1234567890",
          "text": {"body": "Hello"}
        }]
      }
    }]
  }]
}
```

Update webhook to parse Meta's format instead of Twilio's Form data.

Step 5: Update QR Code Delivery (if enabled)

For Meta Cloud API, QR codes must be uploaded first:

```
# 1. Upload QR code to Meta
async with httpx.AsyncClient() as client:
    with open(qr_path, 'rb') as f:
        files = {'file': ('qr.png', f, 'image/png')}
        upload_response = await client.post(
            f"https://graph.facebook.com/v18.0/{META_PHONE_NUMBER_ID}/media",
            headers={"Authorization": f"Bearer {META_ACCESS_TOKEN}"},
            files=files
        )

    media_id = upload_response.json()['id']

# 2. Send message with media_id
await client.post(
    f"https://graph.facebook.com/v18.0/{META_PHONE_NUMBER_ID}/messages",
    headers={
        "Authorization": f"Bearer {META_ACCESS_TOKEN}",
        "Content-Type": "application/json"
    },
    json={
        "messaging_product": "whatsapp",
        "to": phone_number,
        "type": "image",
        "image": {"id": media_id, "caption": response_text}
    }
)
```

Step 6: Configure Meta Webhook

1. Go to Meta Developer Console
2. Navigate to your WhatsApp Business App
3. Configure Webhook:
4. **Callback URL**: `https://whatsapp.your-org.cd/whatsapp/webhook`
5. **Verify Token**: (from your `.env` file)
6. **Webhook Fields**: Select `messages`
7. Click **Verify and Save**

Step 7: Test the Migration

```
# Test Meta webhook verification
curl "https://whatsapp.your-org.cd/whatsapp/webhook?hub.mode=subscribe&hub.verify_token=YOUR_TOKEN&hub.challenge=test123"
# Should return: test123

# Send test message from WhatsApp
# Check logs to verify Meta webhook is working
tail -f logs/app.log
```

Step 8: Remove Twilio (Optional)

Once Meta Cloud API is working:

```
# Remove Twilio-specific code
# Update .env to remove Twilio variables
# Uninstall Twilio SDK (if desired)
pip uninstall twilio
```

Migration Checklist

- [ ] Meta Business Account verified
- [ ] Meta access token obtained
- [ ] Environment variables updated
- [ ] webhook.py updated for Meta API
- [ ] Webhook verification endpoint added
- [ ] Message parsing updated
- [ ] QR code delivery updated (if enabled)
- [ ] Meta webhook configured and verified
- [ ] Test messages sent successfully
- [ ] All features working (UIC generation, QR codes)
- [ ] Twilio credentials removed (optional)

Rollback Plan

If migration fails, rollback to Twilio:

```
# Revert webhook.py changes
git checkout HEAD -- app/api/webhook.py

# Update Twilio webhook URL
# Restart application
sudo systemctl restart whatsapp-uic
```

Troubleshooting

Common Issues and Solutions

Issue 1: WhatsApp Messages Not Received

Symptoms:
- User sends message but bot doesn't respond
- Twilio shows webhook errors

Solutions:
1. Check Twilio webhook configuration:

```
# Verify URL is correct
curl https://whatsapp.your-org.cd/whatsapp/webhook
```

1. Check application logs:

```
sudo journalctl -u whatsapp-uic -n 50
```

1. Verify Nginx is running:

```
sudo systemctl status nginx
```

1. Test webhook manually:

```
curl -X POST https://whatsapp.your-org.cd/whatsapp/webhook \
  -d "From=whatsapp:+1234567890" \
  -d "Body=Hello"
```

Issue 2: Database Connection Errors

Symptoms:
- Application crashes on startup
- Error messages about database connection

Solutions:
1. Check PostgreSQL is running:

```
sudo systemctl status postgresql
```

1. Verify database credentials in `.env`:

```
grep DATABASE_URL .env
```

1. Test database connection:

```
sudo -u postgres psql -d uic_db
```

1. Restart application:

```
sudo systemctl restart whatsapp-uic
```

Issue 3: SSL Certificate Problems

Symptoms:
- Browser shows “Not Secure”
- Webhook fails with SSL errors

Solutions:
1. Verify certificate files exist:

```
sudo ls -l /etc/letsencrypt/live/whatsapp.your-org.cd/
```

1. Test SSL configuration:

```
sudo nginx -t
```

1. Renew certificate if expired:

```
sudo certbot renew
sudo systemctl reload nginx
```

Issue 4: DHIS–2 Integration Failures

Symptoms:
- UIC generated but not registered in DHIS–2
- Errors in logs about API calls

Solutions:
1. Verify DHIS–2 credentials:

```
grep DHIS2_ .env
```

1. Test DHIS-2 connection:

```
curl -u username:password https://your-dhis2-host.org/api/me
```

1. Check DHIS-2 endpoint availability:

```
curl -X POST https://your-dhis2-host.org/api/uic/validate \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"uic": "TEST123M"}'
```

1. Review DHIS-2 specific logs:

```
grep "dhis2" /home/whatsapp-uic-generator/logs/app.log
```

Issue 5: High Memory Usage

Symptoms:
- Server becomes slow
- Application crashes randomly

Solutions:
1. Check memory usage:

```
free -h
htop
```

1. Reduce number of workers in systemd service:

```
sudo nano /etc/systemd/system/whatsapp-uic.service
# Change --workers 4 to --workers 2
sudo systemctl daemon-reload
sudo systemctl restart whatsapp-uic
```

1. Add swap space:

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
```

Support

Getting Help

Documentation

- Full technical documentation: See README files in repository
- API documentation: Visit `https://whatsapp.your-org.cd/docs`
- Configuration guide: See `SETUP.md`

Contact Information

For technical support, please use the repository's GitHub Issues or contact your local technical team.

Reporting Issues

When reporting problems, include:

1. **Error Description**: What were you trying to do?
2. **Error Messages**: Copy exact error text
3. **Logs**: Recent application logs
4. **Environment**: Server specs, OS version

5. **Steps to Reproduce**: How to trigger the error

Example Issue Report:

```
Title: Webhook not receiving messages

Description:
When users send WhatsApp messages, they receive no response.

Error Messages:
[2025-01-17 14:30:22] ERROR: Connection refused to localhost:8000

Logs:
[paste relevant logs here]

Environment:
- Ubuntu 22.04 LTS
- Python 3.12
- 2GB RAM / 2 vCPU

Steps to Reproduce:
1. Send "Hello" to WhatsApp number
2. Wait 30 seconds
3. No response received
```

License

Acknowledgments

- Built with Python FastAPI, Twilio, and love from Canada
- For deployment in the Democratic Republic of the Congo

Last Updated: January 2026
Version: 1.0.0