Email & Phone Validator Telegram Bot

Developer Handover Document

Version 1.0 | August 1, 2025

Email & Phone Validator Telegram Bot - Developer Handover Document

Project Overview

This is a comprehensive Telegram bot that provides bulk email and phone number validation services with a cryptocurrency subscription payment system. The bot validates emails through DNS, MX record, and SMTP connectivity checks, and validates phone numbers using Google's libphonenumber library with carrier detection, country identification, and format validation.

Key Features

- **Dual Validation Services**: Email and phone number validation
- **File Format Support**: CSV, Excel, TXT file processing
- **Real-time Processing**: Concurrent validation with live progress updates
- **Cryptocurrency Payments**: Real BlockBee API integration for 8+ cryptocurrencies
- **Enterprise Scale**: Supports 1000+ concurrent users with rate limiting
- **Mobile-First UI**: Optimized Telegram keyboards and messaging
- **Trial System**: 20,000 free validations before requiring subscription

System Architecture

Core Components

1. Telegram Bot Framework (`bot.py`, `main.py`)

- **Library**: python-telegram-bot v21.7
- **Architecture**: Async/await pattern for concurrent processing
- **Handler Pattern**: Modular handler classes for different functionalities
- **Inline Keyboards**: Rich interactive menus using Telegram's inline keyboard system

2. Database Layer (`database.py`, `models.py`)

- **ORM**: SQLAlchemy with declarative models
- **Database Support**: PostgreSQL (production), SQLite (development)
- **Session Management**: Context-managed database sessions
- **Models**: User, Subscription, ValidationJob with foreign key relationships

3. Validation Engines

Email Validation (`email_validator.py`)

- **Multi-layer Validation**: Syntax, DNS lookup, MX record verification, SMTP connectivity
- **Performance**: 25-email batches with 15-second timeouts
- **SMTP Optimization**: 0.5-second timeouts with optimized handshakes
- **Concurrent Processing**: Thread pool executor with 20 workers per batch
- **Speed**: 15-30 emails/second with real-time progress tracking

Phone Validation (`phone_validator.py`)

- **Library**: Google's libphonenumber (industry standard)
- **Features**: Format validation, country detection, carrier identification
- **International Support**: Handles phone numbers from all countries
- **Smart Extraction**: Pattern matching and phonenumbers library
- **Rich Metadata**: International/national formatting, country info, carrier names

4. File Processing (`file_processor.py`)

- **Formats**: CSV, Excel, TXT
- **Library**: pandas for efficient data processing
- **Security**: File validation, size limits, format verification
- **Management**: Temporary file handling with cleanup

5. Payment System (`services/blockbee_service.py`, `webhook_handler.py`)

- **Provider**: BlockBee API for cryptocurrency payments
- **Currencies**: Bitcoin, Ethereum, USDT (TRC20/ERC20), Litecoin, Dogecoin, TRX, BSC
- **Features**: Real-time conversion, QR code generation, webhook confirmations
- **Architecture**: Flask webhook server + Telegram bot dual setup

Handler Structure

Start Handler (`handlers/start.py`)

- User registration and welcome flow
- Trial system initialization
- Main menu navigation

Validation Handler (`handlers/validation.py`)

- Validation type selection (Email/Phone)
- File upload processing
- Batch validation orchestration
- Results delivery and CSV generation

Subscription Handler (`handlers/subscription.py`)

- Payment flow management
- Cryptocurrency selection
- BlockBee API integration
- Subscription status tracking

Dashboard Handler (`handlers/dashboard.py`)

- Usage statistics display
- Validation history
- Subscription status

Technical Implementation Details

Database Schema

```
-- Users table
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    telegram_id BIGINT UNIQUE NOT NULL,
    username VARCHAR(255),
    first_name VARCHAR(255),
    last_name VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    trial_validations_used INTEGER DEFAULT 0,
    total_validations INTEGER DEFAULT 0
);

-- Subscriptions table
CREATE TABLE subscriptions (
    id INTEGER PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    status VARCHAR(50) DEFAULT 'pending',
    payment_amount_usd DECIMAL(10,2),
    payment_currency_crypto VARCHAR(20),
    payment_address VARCHAR(255),
    payment_amount_crypto DECIMAL(20,8),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    activated_at TIMESTAMP,
    expires_at TIMESTAMP,
    transaction_hash VARCHAR(255)
);

-- Validation jobs table
CREATE TABLE validation_jobs (
    id INTEGER PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    validation_type VARCHAR(20), -- 'email' or 'phone'
    total_count INTEGER,
    valid_count INTEGER DEFAULT 0,
    invalid_count INTEGER DEFAULT 0,
    status VARCHAR(50) DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    completed_at TIMESTAMP,
    results_file_path VARCHAR(500)
);
```

Configuration Management (`config.py`)

```
# Environment Variables Required
TELEGRAM_BOT_TOKEN = os.environ.get('TELEGRAM_BOT_TOKEN')
BLOCKBEE_API_KEY = os.environ.get('BLOCKBEE_API_KEY')
DATABASE_URL = os.environ.get('DATABASE_URL')
BLOCKBEE_WEBHOOK_URL = os.environ.get('BLOCKBEE_WEBHOOK_URL')

# System Configuration
TRIAL_LIMIT = 20000  # Free validations
SUBSCRIPTION_PRICE_USD = 9.99
MAX_FILE_SIZE_MB = 10
BATCH_SIZE_EMAIL = 25
BATCH_SIZE_PHONE = 100
CONCURRENT_WORKERS = 20
```

BlockBee API Integration

Payment Creation Flow

```
# Endpoint: GET https://api.blockbee.io/{currency}/create/
params = {
    'callback': callback_url,
    'apikey': self.api_key,
    'address': receiving_address,
    'convert': 1,
    'pending': 1,  # Notify for pending transactions
    'post': 1,     # Use POST for webhooks
    'json': 1      # JSON format for webhooks
}
```

Webhook Processing

```
# Webhook endpoint: POST /webhook/blockbee
# Processes payment confirmations and activates subscriptions
# Returns 'ok' response required by BlockBee
```

Validation Processing Flow

Email Validation Pipeline

1. **Syntax Check**: Regex pattern validation
2. **DNS Lookup**: Domain existence verification
3. **MX Record Check**: Mail server availability
4. **SMTP Test**: Connection attempt with 0.5s timeout
5. **Result Classification**: Valid/Invalid/Unknown

Phone Validation Pipeline

1. **Text Extraction**: Extract numbers from input text
2. **Format Parsing**: libphonenumber parsing attempt
3. **Validation**: Check if number is valid/possible
4. **Metadata Extraction**: Country, carrier, timezone info
5. **Formatting**: International and national formats

Deployment Configuration

Environment Setup

```
# Required Environment Variables
export TELEGRAM_BOT_TOKEN="your_bot_token"
export BLOCKBEE_API_KEY="your_blockbee_api_key"
export DATABASE_URL="postgresql://user:pass@host:5432/dbname"
export BLOCKBEE_WEBHOOK_URL="https://your-domain.com/webhook/blockbee"

# Install Dependencies
uv add python-telegram-bot sqlalchemy pandas openpyxl dnspython phonenumbers flask requests qrcode pillow
```

Replit Deployment

```
# .replit configuration
run = "python main.py"
modules = ["python-3.11"]

[deployment]
run = ["python", "main.py"]
deploymentTarget = "cloudrun"

[[ports]]
localPort = 5000
externalPort = 80
```

Workflow Configuration

```
# Bot Server Workflow
name: "Bot Server"
command: "python main.py"
wait_for_port: 5000
```

Operational Guidelines

Monitoring and Logging

- **Application Logs**: Comprehensive logging throughout all components
- **Error Tracking**: Exception handling with detailed error messages
- **Performance Metrics**: Real-time speed tracking and ETA calculations
- **Database Monitoring**: Session management and connection pooling

Security Considerations

- **API Key Protection**: Environment variable storage only
- **File Validation**: Size limits and format verification
- **Input Sanitization**: SQL injection prevention via ORM
- **Rate Limiting**: Built into validation processing

Maintenance Tasks

- **Database Cleanup**: Regular cleanup of completed validation jobs
- **File Management**: Temporary file cleanup and storage management
- **Subscription Monitoring**: Track payment confirmations and renewals
- **Performance Tuning**: Monitor concurrent user limits and adjust workers

Troubleshooting Guide

Common Issues

Payment System

- **"Payment service unavailable"**: Check BLOCKBEE_API_KEY validity
- **Webhook not receiving**: Verify BLOCKBEE_WEBHOOK_URL accessibility
- **Address generation fails**: Ensure receiving addresses configured in BlockBee dashboard

Validation Issues

- **Slow email validation**: Adjust BATCH_SIZE_EMAIL and CONCURRENT_WORKERS
- **Phone validation errors**: Verify phonenumbers library installation
- **File processing fails**: Check file size limits and format support

Database Issues

- **Connection errors**: Verify DATABASE_URL format and credentials
- **Migration needs**: Use SQLAlchemy migrations for schema changes
- **Performance**: Monitor connection pooling and session management

Performance Optimization

- **Concurrent Users**: System tested for 1000+ concurrent users
- **Validation Speed**: Email (15-30/sec), Phone (50-100/sec)
- **Memory Management**: Proper session cleanup and file handling
- **Database Indexing**: Ensure proper indexes on frequently queried fields

API References

BlockBee API Endpoints Used

- `GET /{ticker}/create/` - Create payment addresses
- `GET /{ticker}/convert/` - Currency conversion
- `GET /{ticker}/info/` - Ticker information and minimums
- `GET /{ticker}/qrcode/` - QR code generation
- `POST /webhook/blockbee` - Payment confirmation webhooks

Telegram Bot API Features

- Inline keyboards for navigation
- File upload handling
- Message editing for real-time updates
- Callback query processing
- Document sending for results

Future Enhancement Opportunities

Technical Improvements

- **Caching Layer**: Redis for improved performance
- **Message Queue**: Celery for background job processing
- **API Rate Limiting**: More sophisticated rate limiting
- **Database Sharding**: For massive scale deployments

Feature Additions

- **Additional Payment Methods**: More cryptocurrency options
- **Bulk API Access**: REST API for enterprise clients
- **Advanced Analytics**: Detailed validation analytics
- **Multi-language Support**: Internationalization

Monitoring Enhancements

- **Health Checks**: Comprehensive system health monitoring

- **Metrics Dashboard**: Real-time performance dashboard
- **Alert System**: Automated alerting for system issues
- **Usage Analytics**: Detailed user behavior analytics

Code Quality Standards

Development Practices

- **Type Hints**: Python type annotations throughout
- **Error Handling**: Comprehensive exception handling
- **Logging**: Structured logging with appropriate levels
- **Documentation**: Inline comments and docstrings
- **Testing**: Unit tests for critical components

Code Organization

- **Modular Design**: Separated concerns and single responsibility
- **Handler Pattern**: Clean separation of bot functionality
- **Service Layer**: Business logic abstraction
- **Configuration Management**: Environment-based configuration

Contact and Support

Documentation Updates

This document should be updated when: - Major architectural changes are made - New features are added - Configuration changes are required - Performance optimizations are implemented

Key Files to Monitor

- `replit.md` - Project overview and recent changes
- `config.py` - System configuration
- `models.py` - Database schema
- `services/blockbee_service.py` - Payment system
- `handlers/` - Bot functionality

Document Version: 1.0
Last Updated: August 1, 2025
Next Review: Monthly or after major changes

Document Information:

Generated: August 1, 2025 | Version: 1.0 | Format: PDF