

Enterprise Bot Product Factory

FastAPI + ARQ + PostgreSQL + Chatwoot + WhatsApp + Telegram + Dual-Tier AI
Guardrails

Repository: <https://github.com/sh15aa/botserusable>

Author Account: foundershivanand@gmail.com

Security Level: Zero-Vulnerability Audited

Date: October 2026

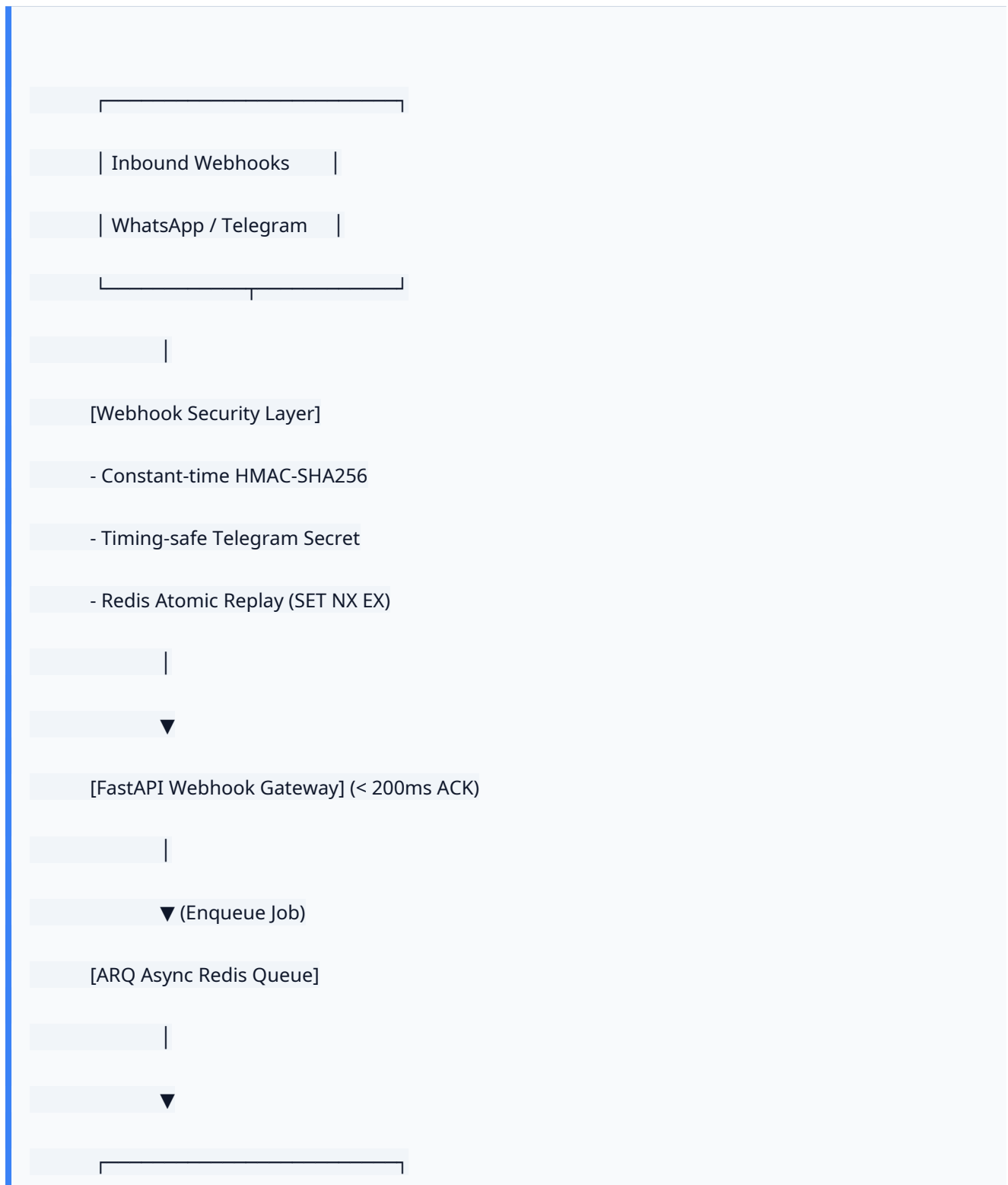


Enterprise Bot Product Factory (Python: FastAPI + ARQ + Chatwoot)

A reusable, production-grade **Bot Product Factory** built with **FastAPI**, **ARQ (Async Redis Queue)**, **PostgreSQL (AsyncPG)**, and **Chatwoot**.

It provides a channel-independent foundation for WhatsApp Cloud API and Telegram Bot API, with human handoff, dual-tier AI guardrails, distributed concurrency locks, and zero-vulnerability security controls.

Architecture & Topology



| ARQ Background Worker |

|

|

|

▼ ▼ ▼

[Rate Limiter] [Loop Detector] [Distributed Lock]

(Redis Lua) (Rolling SHA) (Redis Mutex)

|

▼

[Conversation Engine & FSM]

|

|

|

[Human Handoff?] [AI Assistant]

|

▼ ▼

[Chatwoot Desk] [Dual-Tier AI Guardrail]

- Sync Contacts - PII Redaction

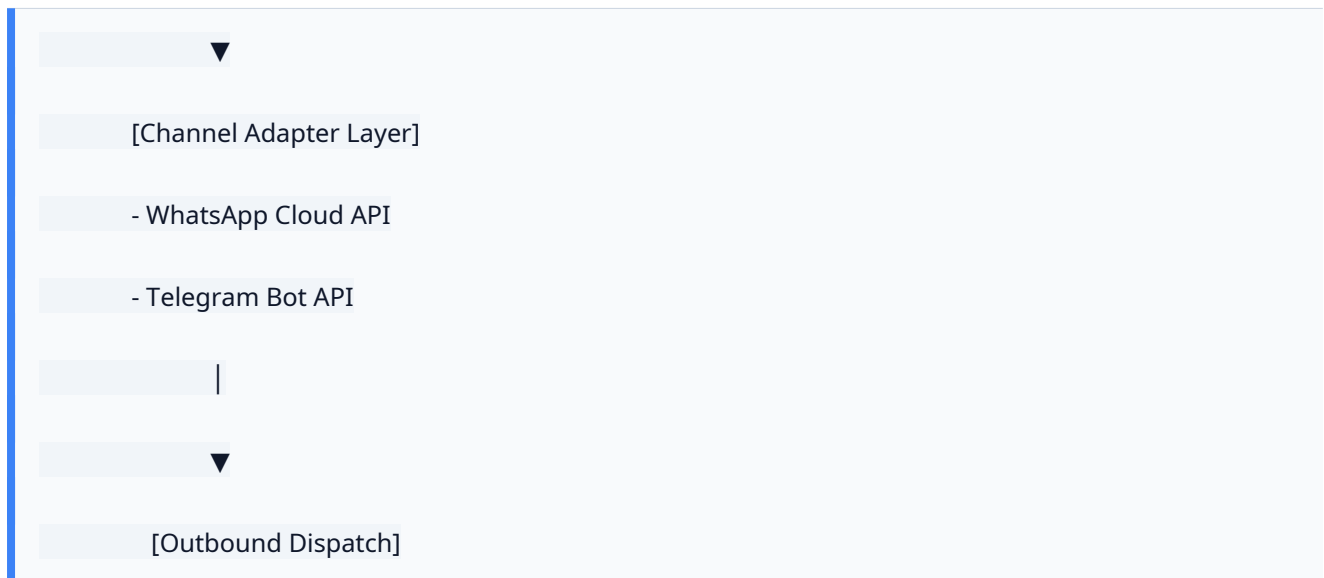
- Sync Messages - Delimiter Sandboxing

- Bi-directional Relay - Dynamic Canary Tokens

|

|

|



Zero-Vulnerability Security Checklist

Threat Vector	Mitigation Strategy	Implementation Location
Side-Channel Timing Attacks	Constant-time comparison with <code>`hmac.compare_digest`</code>	[security.py](file:///home/shiva/botsreusable/app/gateway/security.py)
Replay & Duplicate Events	Atomic Redis <code>`SET key 1 EX 86400 NX`</code>	[security.py](file:///home/shiva/botsreusable/app/gateway/security.py)
Rapid Button Mashing / Concurrency Race	Distributed lock per user session via Redis mutex + Lua release	[fsm.py](file:///home/shiva/botsreusable/app/engine/fsm.py)

Prompt Injection & System Prompt Leaking	Input delimiter sandboxing (``) + Dynamic canary token leak detection	[guardrails.py](file:///home/shiva/botsreusable/app/ai/guardrails.py)
PII Data Exposure	Automated regex masking for credit cards, emails, SSNs, and phone numbers	[guardrails.py](file:///home/shiva/botsreusable/app/ai/guardrails.py)
Denial of Service / Flooding	Atomic sliding-window rate limiting via Redis Lua script	[rate_limiter.py](file:///home/shiva/botsreusable/app/security/rate_limiter.py)
Bot-to-Bot Ping-Pong Loops	Rolling SHA-256 hash tracking over sliding observation window	[loop_detector.py](file:///home/shiva/botsreusable/app/security/loop_detector.py)

Quickstart & Deployment

1. Environment Setup

Copy `.env.example` to `.env` and fill in your secrets:

```
cp .env.example .env
```

2. Run with Docker Compose

```
docker-compose up -d --build
```

This starts:

- **API Server:** `http://localhost:8000`
- **ARQ Worker:** Background queue runner
- **Redis:** `localhost:6379`
- **PostgreSQL:** `localhost:5432`



Provider Webhook Setup

WhatsApp Cloud API (Meta for Developers)

1. Set Callback URL: `https://yourdomain.com/webhooks/whatsapp`
2. Verify Token: Set to match `WHATSAPP_VERIFY_TOKEN` in `.env`.
3. Webhook fields to subscribe: `messages`.
4. App Secret: Set `WHATSAPP_APP_SECRET` to enable HMAC-SHA256 signature verification.

Telegram Bot API

1. Set your webhook with your secret token:

```
curl -X POST "https://api.telegram.org/bot<YOUR_BOT_TOKEN>/setWebhook" \  
-H "Content-Type: application/json" \  
-d '{"url": "https://yourdomain.com/webhooks/telegram", "secret_token": "  
<YOUR_TELEGRAM_SECRET_TOKEN>"}'
```

Chatwoot (Live Agent Desk)

1. In Chatwoot, navigate to **Settings** -> **Integrations** -> **Webhooks**.

2. Add Webhook URL: <https://yourdomain.com/webhooks/chatwoot>

3. Subscribed events:

- `message_created`

- `conversation_status_changed`

1. Set `CHATWOOT_WEBHOOK_SECRET` in `.env`.

2. Now, when a user triggers handoff (e.g. clicks "Talk to Agent" or types "human"):

- The user is transitioned to `HUMAN_HANDOFF`.

- A contact and conversation are created in Chatwoot.

- Any agent replies inside Chatwoot are routed directly back to the user's WhatsApp or Telegram!

- When the agent marks the conversation as **Resolved**, the bot automatically resumes control.