GREEDYGAME

AI Ticket Responder — Workflow Overview

Professional summary of architecture, tech stack, and guided user experience.

Summary

- Guided support for offer-related queries with FAQ-driven answers and guard rails.
- Web UI connects via WebSocket to a FastAPI backend; responses stream from a local LLM with RAG context.
- Features include status-aware guidance, inactivity nudges, domain guard, input validation, rate limiting, content filtering, and expired-offer recommendations.

Tech Stack

- Frontend: HTML/CSS/JavaScript, WebSocket client
- Backend: FastAPI (HTTP + WebSocket)
- Knowledge: FAQs (data/faqs.txt) embedded with Chroma (rag.py)
- LLM: Ollama HTTP API streaming (llm.py), model via env vars
- Business logic: offer status handling (offer_logic.py), mock API (mock_offer_api.py)
- Guard rails: validation, rate limiting, content filtering, domain guard (guard_rails.py)

Core Components

- UI: Header with offer title and status badge; message bubbles; contextual suggestion buttons; input row revealed only when user opts to type.
- Backend: WebSocket /ws streams bot output; HTTP /chat kept for compatibility; background inactivity monitor sends 30s nudge.
- RAG + LLM: rag.search_docs retrieves FAQ context; llm.ask_llm streams text; prompt enforces concise, professional tone.
- Offer Logic: Status-aware queries and quick recommendations for expired offers.
- Guard Rails: Input validation, per-IP rate limiting, content filtering, and domain guard to avoid off-topic replies.

End-to-End Flow (Typed Issue)

- User selects offer; header shows title + status badge; status-specific guidance appears.
- User chooses “I’d rather type out my issue”; input row appears.
- Message is sent over WebSocket with offer_id; UI shows user bubble then creates bot bubble.
- process_chat: validation → filtering → rate limiting → domain guard → offer context → RAG → LLM streaming.
- UI fills the bot bubble as chunks stream; backend sends a delimiter to close the bubble.

- Expired offers append a brief list of quick alternative offers.
- Idle for 30s, the backend sends a professional nudge.

Guided Button Flow (No Typing)

- Ongoing/Completed: “I’m not able to complete the offer” → Help Options: Watch 2■min guide (CTA) or Step■by■step instructions (bullets); then browse offers or report issue.
- Expired: “Check out other available offers” or “Report this issue to us”.
- Reporting opens mailto:support@greedygame.com with prefilled subject/body; transcript records selection as user bubble.
- Browse other offers displays quick alternatives from the dropdown; selection switches offer context and resets guidance.
- End Chat closes the flow with a thank■you and hides inputs.

Connections Overview

- UI ↔ Backend: WebSocket streaming for live replies; HTTP is retained as backup.
- Backend ↔ RAG: rag.search_docs uses embedded FAQs; prompt merges retrieved context.
- Backend ↔ LLM: llm.ask_llm streams text from Ollama; errors are surfaced as text.
- Backend Guard Rails: Pre■LLM checks gate requests; post■LLM filtering sanitizes final text.

Operational Notes

- Start backend: uvicorn main:app --reload --port 8000 (from backend/ within venv).
- Configure LLM: OLLAMA_URL and MODEL_NAME env vars if non■default.
- Update FAQs: edit data/faqs.txt and restart; RAG collection resets on startup.
- Security: no secrets in repo; avoid sending PII; domain guard deflects non■offer topics.

Key Files

- Backend entry: /Users/rajdeeproy/ai_ticket_responder/backend/main.py
- UI: /Users/rajdeeproy/ai_ticket_responder/ui/index.html
- RAG: /Users/rajdeeproy/ai_ticket_responder/backend/rag.py
- LLM client: /Users/rajdeeproy/ai_ticket_responder/backend/llm.py
- Offer logic: /Users/rajdeeproy/ai_ticket_responder/backend/offer_logic.py
- Mock offers: /Users/rajdeeproy/ai_ticket_responder/backend/mock_offer_api.py
- Guard rails: /Users/rajdeeproy/ai_ticket_responder/backend/guard_rails.py
- FAQs: /Users/rajdeeproy/ai_ticket_responder/data/faqs.txt