📘 Manual de Expansão Agêntica: E-mail e SaaS (Ambiente de Testes)

Este documento descreve a arquitetura e os passos necessários para transformar o **Bot RAG** (apenas consulta) em um **Agente Autônomo** (execução de tarefas), utilizando um ambiente seguro de homologação ("Sandbox").

🏗️ Parte 1: O Conceito (Laboratório de Segurança)

Para desenvolver funcionalidades críticas como envio de e-mails e manipulação de dados de clientes (CRM/SaaS), não devemos conectar a IA diretamente aos serviços de produção (Gmail real, Salesforce real) durante a fase de desenvolvimento.

Utilizamos o conceito de **Mocking** (Simulação) e **Staging** (Ambiente de Palco). Isso garante que, se a IA "alucinar" ou entrar em loop, nenhum cliente real será afetado.

Arquitetura de Testes

🛠️ Parte 2: Infraestrutura (As Ferramentas)

Utilizamos dois serviços gratuitos que simulam perfeitamente o comportamento de sistemas reais:

1. Mailtrap (Simulador de SMTP)

O Mailtrap atua como um servidor de e-mail "buraco negro". Ele aceita conexões SMTP reais, mas **não entrega** a mensagem para o destinatário final. O e-mail fica retido em uma caixa de entrada virtual que apenas você acessa.

- **Uso:** Testar formatação, envio e lógica da IA.
- **Dados Necessários:** `Host`, `Port`, `Username`, `Password`.
- **Onde obter:** mailtrap.io > Email Testing > Inboxes.

2. MockAPI (Simulador de Backend/SaaS)

O MockAPI permite criar APIs REST completas (GET, POST, PUT, DELETE) em segundos, simulando um banco de dados.

- **Uso:** Simular criação de tickets de suporte, cadastro de clientes ou consulta de status.
- **Recurso Criado:** `/tickets` (com campos `id`, `titulo`, `descricao`, `status`).
- **Onde obter:** mockapi.io > Projects > New Resource.

💻 Parte 3: Implementação (O Código)

A implementação ocorre em duas etapas: definição das ferramentas (`src/tools.py`) e entrega ao cérebro (`src/services/rag_service.py`).

A. Criando as Ferramentas (`src/tools.py`)

```python
import smtplib
import requests
from email.mime.text import MIMEText
from langchain_core.tools import tool

# ⚙️ CONFIGURAÇÕES (Preencher com dados do Mailtrap/MockAPI)
SMTP_HOST = "sandbox.smtp.mailtrap.io"
SMTP_PORT = 2525
SMTP_USER = "SEU_USER_MAILTRAP"
SMTP_PASS = "SEU_PASS_MAILTRAP"
MOCK_API_URL = "https://SEU_ID.mockapi.io/api/v1"

# --- FERRAMENTA 1: Envio de E-mail (SMTP Real/Sandbox) ---
@tool
def enviar_email_real(destinatario: str, assunto: str, corpo: str):
    """
    Envia um e-mail REAL usando protocolo SMTP.
    Use para notificar suporte, enviar relatórios ou contatar o usuário.
    """
    try:
        msg = MIMEText(corpo)
        msg['Subject'] = assunto
        msg['From'] = "bot@uema.br"
        msg['To'] = destinatario

        with smtplib.SMTP(SMTP_HOST, SMTP_PORT) as server:
            server.login(SMTP_USER, SMTP_PASS)
            server.sendmail("bot@uema.br", [destinatario], msg.as_string())

        return f"✅ E-mail enviado com sucesso para {destinatario} (Verifique o Mailtrap)!"
    except Exception as e:
        return f"❌ Falha ao enviar e-mail: {str(e)}"

# --- FERRAMENTA 2: Abrir Chamado (SaaS) ---
@tool
def abrir_ticket_suporte(titulo: str, descricao: str):
    """
    Abre um chamado técnico no sistema de SaaS (CRM).
    Retorna o ID e status do chamado criado.
    """
    url = f"{MOCK_API_URL}/tickets"
```

```
    payload = {"titulo": titulo, "descricao": descricao, "status": "ABERTO"}

    try:
        response = requests.post(url, json=payload)
        if response.status_code == 201:
            dados = response.json()
            return f"✅ Chamado criado! ID: {dados['id']} | Status: {dados['status']}"
        return f"❌ Erro na API: {response.status_code}"
    except Exception as e:
        return f"❌ Erro de conexão: {str(e)}"

# --- FERRAMENTA 3: Consultar Chamados (SaaS) ---
@tool
def consultar_meus_tickets():
    """Consulta a lista de chamados abertos no SaaS."""
    try:
        response = requests.get(f"{MOCK_API_URL}/tickets")
        tickets = response.json()
        if not tickets: return "Nenhum chamado encontrado."

        resumo = "\n".join([f"- ID {t['id']}: {t['titulo']} ({t['status']})" for t in tickets])
        return f"📋 Chamados encontrados:\n{resumo}"
    except Exception as e:
        return f"Erro ao ler SaaS: {str(e)}"
```

B. Conectando à IA (`src/services/rag_service.py`)

No método `inicializar` :

```
def inicializar(self):
    # Importar as novas ferramentas
    from src.tools import tool_pdf, abrir_chamado_glpi, consultar_fila, \
                          enviar_email_real, abrir_ticket_suporte, consultar_meus_tickets

    # Adicionar à lista de capacidades da IA
    tools = [
        tool_pdf,
        abrir_chamado_glpi,
        consultar_fila,
        enviar_email_real,      # Nova Capacidade: E-mail
        abrir_ticket_suporte,   # Nova Capacidade: Criar no CRM
        consultar_meus_tickets  # Nova Capacidade: Ler do CRM
    ]

    # Vincular ao modelo Llama
    llm = ChatGroq(...).bind_tools(tools)

    # ... restante do código
```

🧪 Parte 4: Homologação (O Teste Prático)

Após reiniciar o bot (`docker compose restart bot`), execute os seguintes cenários no WhatsApp para validar a autonomia do Agente.

✅ Teste A: E-mail

1. **Comando:** "Mande um e-mail para o diretor avisando que o sistema caiu."
2. **Ação Esperada:** O Bot confirma o envio no WhatsApp.
3. **Verificação:** Abra o **Mailtrap.io**. O e-mail deve aparecer na sua Inbox com o assunto e corpo criados pela IA.

✅ Teste B: SaaS (Escrita - POST)

1. **Comando:** "Abra um chamado técnico: a impressora do laboratório está sem tinta."
2. **Ação Esperada:** O Bot responde "Chamado criado! ID: X".
3. **Verificação:** Abra o **MockAPI.io** > Resource `tickets` > Aba Data. O novo registro deve estar lá.

✅ Teste C: SaaS (Leitura - GET)

1. **Comando:** "Quais chamados estão abertos?"
2. **Ação Esperada:** O Bot vai até o MockAPI, lê a lista de tickets (incluindo o que você criou no Teste B) e resume para você no chat.