

Guia Completo: Chatbot WhatsApp para Clínica com GestãoDS

Índice

1. [Visão Geral do Projeto](#)
 2. [Arquitetura Detalhada](#)
 3. [Configuração Z-API](#)
 4. [Estrutura do Projeto](#)
 5. [Implementação Passo a Passo](#)
 6. [Integração com GestãoDS](#)
 7. [Deploy e Testes](#)
 8. [Cronograma Detalhado](#)
-

Visão Geral do Projeto {#visão-geral}

Objetivo

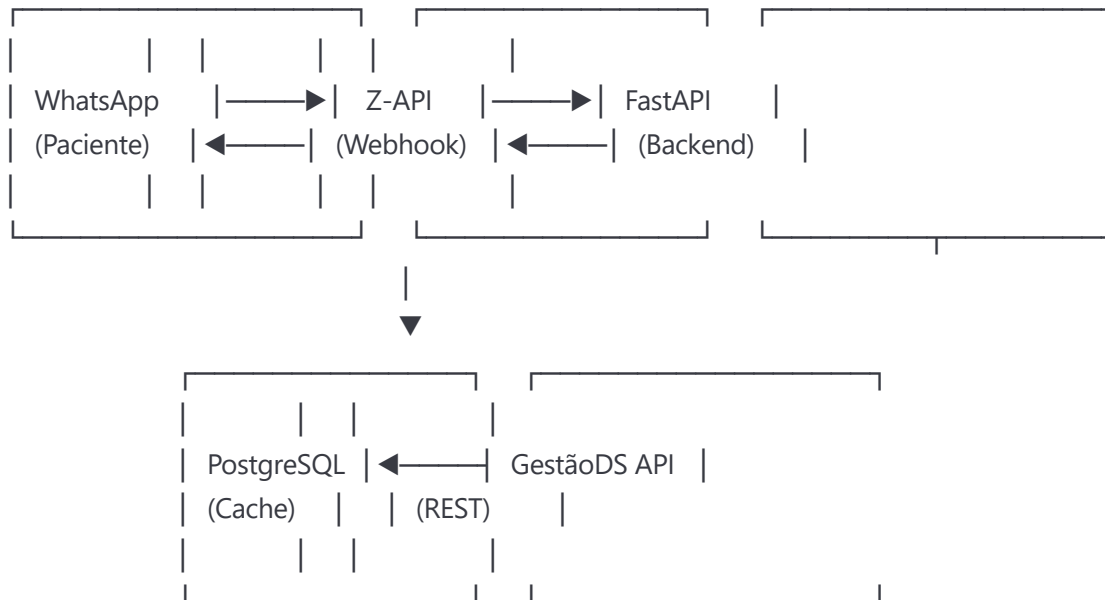
Criar um assistente virtual no WhatsApp que automatize o atendimento da Clínica Gabriela Nassif, integrado com o sistema GestãoDS para:

- Agendar consultas automaticamente
- Enviar lembretes 24h antes
- Gerenciar cancelamentos
- Manter lista de espera inteligente

Stack Tecnológica Definitiva

- **Backend:** Python 3.11 + FastAPI
 - **WhatsApp:** Z-API (sem necessidade de aprovação Meta)
 - **Banco de Dados:** PostgreSQL (para cache e estado das conversas)
 - **Deploy:** Railway ou Render
 - **Agendador:** APScheduler
 - **HTTP Client:** httpx (assíncrono)
-

Arquitetura Detalhada {#arquitetura}



Fluxo de Dados

1. **Mensagem Recebida:** WhatsApp → Z-API → Webhook FastAPI
2. **Processamento:** FastAPI analisa contexto → Consulta GestãoDS
3. **Resposta:** FastAPI → Z-API → WhatsApp
4. **Agendamento:** Dados salvos no GestãoDS + Cache local

🔧 Configuração Z-API {#configuração-z-api}

1. Criar Conta Z-API

1. Acesse <https://app.z-api.io/>
2. Clique em "Crie uma conta grátis"
3. Complete o cadastro com dados da clínica

2. Configurar Instância WhatsApp

1. No painel Z-API, clique em "Nova Instância"
2. Escolha um nome (ex: "clínica-gabriela")
3. Conecte o WhatsApp:
 - Abra WhatsApp no celular
 - Vá em Configurações → Dispositivos conectados
 - Escaneie o QR Code

3. Configurar Webhooks

No painel da instância:

1. Clique em "Editar" → "Webhooks"

2. Configure as URLs:

- **Webhook de Mensagens Recebidas:** `https://seu-dominio.com/webhook/message`
- **Webhook de Status:** `https://seu-dominio.com/webhook/status`
- **Webhook de Conexão:** `https://seu-dominio.com/webhook/connected`

4. Obter Credenciais

1. Na aba "Segurança", copie:

- **Instance ID:** Ex: `3B8A9C2D-1234-5678-90AB-CDEF12345678`
- **Token:** Ex: `$2b$10$abcdefghijklmnopqrstuvwxyz123456`
- **Client Token:** Para autenticação adicional

5. Endpoints Z-API Principais

```
python

# Base URL
BASE_URL = "https://api.z-api.io/instances/{instance_id}/token/{token}"

# Endpoints principais:
# - POST /send-text      # Enviar mensagem de texto
# - POST /send-button-list # Enviar botões de opção
# - POST /send-link      # Enviar link
# - GET /status          # Verificar status da instância
# - POST /read-message   # Marcar como lida
```

Estrutura do Projeto {#estrutura-projeto}

```

chatbot-clinica/
├── app/
│   ├── __init__.py
│   ├── main.py          # FastAPI principal
│   ├── config.py        # Configurações e variáveis
│   ├── models/
│   │   ├── __init__.py
│   │   ├── database.py  # Modelos do banco
│   │   └── schemas.py   # Schemas Pydantic
│   ├── services/
│   │   ├── __init__.py
│   │   ├── whatsapp.py  # Integração Z-API
│   │   ├── gestaods.py  # Integração GestãoDS
│   │   └── conversation.py # Gerenciamento de conversas
│   ├── handlers/
│   │   ├── __init__.py
│   │   ├── webhook.py   # Handlers dos webhooks
│   │   └── messages.py  # Processamento de mensagens
│   ├── utils/
│   │   ├── __init__.py
│   │   ├── validators.py # Validações (CPF, data, etc)
│   │   └── formatters.py # Formatações
│   └── tasks/
│       ├── __init__.py
│       └── reminders.py  # Tarefas agendadas
├── tests/
│   ├── __init__.py
│   ├── test_whatsapp.py
│   ├── test_gestaods.py
│   └── test_conversation.py
├── docker-compose.yml
├── Dockerfile
├── requirements.txt
├── .env.example
├── README.md
└── setup.py

```



Implementação Passo a Passo {#implementação}

Dia 1 - Quarta-feira: Setup e Integrações Base

1. Configuração Inicial (1h)

No Cursor IDE:

1. Criar novo projeto: `Cmd/Ctrl + Shift + N`

2. Abrir terminal integrado: `Cmd/Ctrl + J`

3. Configurar ambiente Python:

```
bash

# Criar ambiente virtual
python -m venv venv

# Ativar ambiente (Windows)
venv\Scripts\activate

# Ativar ambiente (Mac/Linux)
source venv/bin/activate

# Instalar dependências
pip install fastapi uvicorn httpx python-dotenv sqlalchemy
pip install apscheduler pydantic redis pytest pytest-asyncio
pip install asyncpg psycopg2-binary alembic
```

2. Arquivo de Configuração (.env)

```
env
```

Z-API Credentials

ZAPI_INSTANCE_ID=3B8A9C2D-1234-5678-90AB-CDEF12345678

ZAPI_TOKEN=\$2b\$10\$abcdefghijklmnopqrstuvwxyz123456

ZAPI_CLIENT_TOKEN=seu_client_token_aqui

GestãoDS API

GESTAODS_API_URL=https://apidev.gestaods.com.br

GESTAODS_TOKEN=seu_token_gestaods

Database

DATABASE_URL=postgresql://user:password@localhost:5432/chatbot_clinica

App Settings

APP_HOST=0.0.0.0

APP_PORT=8000

ENVIRONMENT=development

DEBUG=True

Clinic Info

CLINIC_NAME=Clínica Gabriela Nassif

CLINIC_PHONE=5531999999999

REMINDER_HOUR=18

REMINDER_MINUTE=0

3. Configuração Base (config.py)

```
python
```

```
from pydantic_settings import BaseSettings
from functools import lru_cache
import os

class Settings(BaseSettings):
    # Z-API
    zapi_instance_id: str
    zapi_token: str
    zapi_client_token: str
    zapi_base_url: str = "https://api.z-api.io"

    # GestãoDS
    gestaods_api_url: str
    gestaods_token: str

    # Database
    database_url: str

    # App
    app_host: str = "0.0.0.0"
    app_port: int = 8000
    environment: str = "development"
    debug: bool = True

    # Clinic
    clinic_name: str
    clinic_phone: str
    reminder_hour: int = 18
    reminder_minute: int = 0

    class Config:
        env_file = ".env"
        case_sensitive = False

    @lru_cache()
    def get_settings():
        return Settings()

settings = get_settings()
```

4. Modelos de Banco de Dados (models/database.py)

```
python
```

```
from sqlalchemy import create_engine, Column, String, DateTime, JSON, Boolean, Integer
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from datetime import datetime
import uuid
```

```
Base = declarative_base()
```

```
class Conversation(Base):
```

```
    __tablename__ = "conversations"
```

```
    id = Column(String, primary_key=True, default=lambda: str(uuid.uuid4()))
```

```
    phone = Column(String, nullable=False, index=True)
```

```
    state = Column(String, default="inicio")
```

```
    context = Column(JSON, default={})
```

```
    created_at = Column(DateTime, default=datetime.utcnow)
```

```
    updated_at = Column(DateTime, default=datetime.utcnow, onupdate=datetime.utcnow)
```

```
class Appointment(Base):
```

```
    __tablename__ = "appointments"
```

```
    id = Column(String, primary_key=True, default=lambda: str(uuid.uuid4()))
```

```
    patient_id = Column(String, nullable=False)
```

```
    patient_name = Column(String)
```

```
    patient_phone = Column(String)
```

```
    appointment_date = Column(DateTime)
```

```
    appointment_type = Column(String)
```

```
    status = Column(String, default="scheduled")
```

```
    reminder_sent = Column(Boolean, default=False)
```

```
    created_at = Column(DateTime, default=datetime.utcnow)
```

```
class WaitingList(Base):
```

```
    __tablename__ = "waiting_list"
```

```
    id = Column(String, primary_key=True, default=lambda: str(uuid.uuid4()))
```

```
    patient_id = Column(String, nullable=False)
```

```
    patient_name = Column(String)
```

```
    patient_phone = Column(String)
```

```
    preferred_dates = Column(JSON)
```

```
    priority = Column(Integer, default=0)
```

```
    created_at = Column(DateTime, default=datetime.utcnow)
```

```
    notified = Column(Boolean, default=False)
```

```
# Database setup
```

```
engine = create_engine(settings.database_url)
```

```
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```



```
Base.metadata.create_all(bind=engine)
```

```
def get_db():  
    db = SessionLocal()  
    try:  
        yield db  
    finally:  
        db.close()
```

5. Integração Z-API (services/whatsapp.py)

```
python
```

```
import httpx
from typing import Optional, List, Dict
from app.config import settings
import logging

logger = logging.getLogger(__name__)

class WhatsAppService:
    def __init__(self):
        self.base_url = f"{settings.zapi_base_url}/instances/{settings.zapi_instance_id}/token/{settings.zapi_token}"
        self.headers = {
            "Client-Token": settings.zapi_client_token,
            "Content-Type": "application/json"
        }

    async def send_text(self, phone: str, message: str, delay_message: int = 2):
        """Envia mensagem de texto simples"""
        try:
            async with httpx.AsyncClient() as client:
                payload = {
                    "phone": self._format_phone(phone),
                    "message": message,
                    "delayMessage": delay_message
                }

                response = await client.post(
                    f"{self.base_url}/send-text",
                    json=payload,
                    headers=self.headers
                )

                if response.status_code == 200:
                    logger.info(f"Mensagem enviada para {phone}")
                    return response.json()
                else:
                    logger.error(f"Erro ao enviar mensagem: {response.text}")
                    return None

        except Exception as e:
            logger.error(f"Erro na comunicação com Z-API: {str(e)}")
            return None

    async def send_button_list(self, phone: str, message: str,
                               buttons: List[Dict[str, str]],
                               title: str = "Opções"):
        """Envia mensagem com lista de botões"""
```

try:

```
async with httpx.AsyncClient() as client:
```

```
    payload = {  
        "phone": self._format_phone(phone),  
        "message": message,  
        "buttonList": {  
            "buttons": buttons  
        },  
        "title": title,  
        "footer": settings.clinic_name  
    }
```

```
    response = await client.post(  
        f"{self.base_url}/send-button-list",  
        json=payload,  
        headers=self.headers  
    )
```

```
    return response.json() if response.status_code == 200 else None
```

except Exception as e:

```
    logger.error(f"Erro ao enviar botões: {str(e)}")  
    return None
```

```
async def send_link(self, phone: str, message: str, link: str):
```

```
    """Envia mensagem com link"""
```

try:

```
async with httpx.AsyncClient() as client:
```

```
    payload = {  
        "phone": self._format_phone(phone),  
        "message": message,  
        "linkUrl": link,  
        "title": "Clique aqui",  
        "linkDescription": "Acesse o link para mais informações"  
    }
```

```
    response = await client.post(  
        f"{self.base_url}/send-link",  
        json=payload,  
        headers=self.headers  
    )
```

```
    return response.json() if response.status_code == 200 else None
```

except Exception as e:

```
    logger.error(f"Erro ao enviar link: {str(e)}")  
    return None
```

```

async def mark_as_read(self, phone: str, message_id: str):
    """Marca mensagem como lida"""
    try:
        async with httpx.AsyncClient() as client:
            payload = {
                "phone": self._format_phone(phone),
                "messageId": message_id
            }

            await client.post(
                f"{self.base_url}/read-message",
                json=payload,
                headers=self.headers
            )

    except Exception as e:
        logger.error(f"Erro ao marcar como lida: {str(e)}")

def _format_phone(self, phone: str) -> str:
    """Formata número de telefone para padrão Z-API"""
    # Remove caracteres não numéricos
    phone = ''.join(filter(str.isdigit, phone))

    # Adiciona código do país se não tiver
    if not phone.startswith('55'):
        phone = '55' + phone

    # Adiciona 9 se for celular e não tiver
    if len(phone) == 12 and phone[4] != '9':
        phone = phone[:4] + '9' + phone[4:]

    return phone + '@c.us'

```

6. Integração GestãoDS (services/gestaods.py)

python

```

import httpx
from typing import Optional, List, Dict
from datetime import datetime, timedelta
from app.config import settings
import logging

logger = logging.getLogger(__name__)

class GestaoDS:
    def __init__(self):
        self.base_url = settings.gestaods_api_url
        self.token = settings.gestaods_token
        self.headers = {
            "Authorization": f"Bearer {self.token}",
            "Content-Type": "application/json"
        }

    async def buscar_paciente_cpf(self, cpf: str) -> Optional[Dict]:
        """Busca paciente por CPF"""
        try:
            cpf_limpo = ''.join(filter(str.isdigit, cpf))

            async with httpx.AsyncClient() as client:
                response = await client.get(
                    f"{self.base_url}/api/paciente/{self.token}/{cpf_limpo}/",
                    headers=self.headers,
                    timeout=30.0
                )

            if response.status_code == 200:
                return response.json()
            elif response.status_code == 404:
                logger.info(f"Paciente não encontrado: {cpf}")
                return None
            else:
                logger.error(f"Erro ao buscar paciente: {response.status_code}")
                return None

        except Exception as e:
            logger.error(f"Erro na API GestãoDS: {str(e)}")
            return None

    async def listar_horarios_disponiveis(self,
        data_inicio: datetime,
        data_fim: datetime,
        tipo_consulta: str = "consulta") -> List[Dict]:

```

```
"""Lista horários disponíveis para agendamento"""
```

```
try:
```

```
    async with httpx.AsyncClient() as client:
```

```
        params = {
```

```
            "data_inicio": data_inicio.strftime("%Y-%m-%d"),
```

```
            "data_fim": data_fim.strftime("%Y-%m-%d"),
```

```
            "tipo": tipo_consulta,
```

```
            "disponivel": True
```

```
        }
```

```
    response = await client.get(
```

```
        f"{self.base_url}/api/agenda/horarios-disponiveis/",
```

```
        headers=self.headers,
```

```
        params=params,
```

```
        timeout=30.0
```

```
    )
```

```
    if response.status_code == 200:
```

```
        horarios = response.json()
```

```
        # Formatar horários para exibição
```

```
        return self._formatar_horarios(horarios)
```

```
    else:
```

```
        logger.error(f"Erro ao buscar horários: {response.status_code}")
```

```
        return []
```

```
except Exception as e:
```

```
    logger.error(f"Erro ao listar horários: {str(e)}")
```

```
    return []
```

```
async def criar_agendamento(self,
```

```
    paciente_id: int,
```

```
    data_hora: datetime,
```

```
    tipo: str = "consulta",
```

```
    observacoes: str = "") -> Optional[Dict]:
```

```
"""Cria novo agendamento"""
```

```
try:
```

```
    async with httpx.AsyncClient() as client:
```

```
        payload = {
```

```
            "paciente_id": paciente_id,
```

```
            "data_hora": data_hora.isoformat(),
```

```
            "tipo": tipo,
```

```
            "status": "agendado",
```

```
            "observacoes": observacoes,
```

```
            "origem": "whatsapp_bot"
```

```
        }
```

```
    response = await client.post(
```

```
        f"{self.base_url}/api/agendamento/",
        headers=self.headers,
        json=payload,
        timeout=30.0
    )
```

```
    if response.status_code in [200, 201]:
        return response.json()
    else:
        logger.error(f"Erro ao criar agendamento: {response.text}")
        return None
```

```
except Exception as e:
    logger.error(f"Erro ao agendar: {str(e)}")
    return None
```

```
async def listar_agendamentos_paciente(self, paciente_id: int) -> List[Dict]:
```

```
    """Lista agendamentos do paciente"""
```

```
    try:
```

```
        async with httpx.AsyncClient() as client:
            response = await client.get(
                f"{self.base_url}/api/paciente/{paciente_id}/agendamentos/",
                headers=self.headers,
                timeout=30.0
            )
```

```
            if response.status_code == 200:
                return response.json()
            else:
                return []
```

```
except Exception as e:
    logger.error(f"Erro ao listar agendamentos: {str(e)}")
    return []
```

```
async def cancelar_agendamento(self, agendamento_id: int, motivo: str = "") -> bool:
```

```
    """Cancela agendamento"""
```

```
    try:
```

```
        async with httpx.AsyncClient() as client:
            payload = {
                "status": "cancelado",
                "motivo_cancelamento": motivo,
                "cancelado_por": "paciente_whatsapp"
            }
```

```
            response = await client.patch(
                f"{self.base_url}/api/agendamento/{agendamento_id}/",
```

```
headers=self.headers,  
json=payload,  
timeout=30.0  
)  
  
return response.status_code in [200, 204]
```

```
except Exception as e:  
    logger.error(f"Erro ao cancelar: {str(e)}")  
    return False
```

```
async def buscar_agendamentos_dia(self, data: datetime) -> List[Dict]:
```

```
    """Busca todos agendamentos de um dia específico"""
```

```
    try:
```

```
        async with httpx.AsyncClient() as client:  
            params = {  
                "data": data.strftime("%Y-%m-%d"),  
                "status": "agendado"  
            }
```

```
            response = await client.get(  
                f"{self.base_url}/api/agenda/dia/",  
                headers=self.headers,  
                params=params,  
                timeout=30.0  
            )
```

```
            if response.status_code == 200:  
                return response.json()  
            else:  
                return []
```

```
except Exception as e:  
    logger.error(f"Erro ao buscar agenda do dia: {str(e)}")  
    return []
```

```
def _formatar_horarios(self, horarios: List[Dict]) -> List[Dict]:
```

```
    """Formata horários para exibição amigável"""
```

```
    horarios_formatados = []
```

```
    for horario in horarios:
```

```
        data_hora = datetime.fromisoformat(horario['data_hora'])  
        horarios_formatados.append({  
            'id': horario.get('id'),  
            'data': data_hora.strftime("%d/%m/%Y"),  
            'hora': data_hora.strftime("%H:%M"),  
            'dia_semana': self._get_dia_semana(data_hora),
```



```
        'tipo': horario.get('tipo', 'consulta'),
        'profissional': horario.get('profissional', 'Dr(a). Gabriela Nassif')
    })

    return horarios_formatados

def _get_dia_semana(self, data: datetime) -> str:
    """Retorna dia da semana em português"""
    dias = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado', 'Domingo']
    return dias[data.weekday()]
```

Dia 2 - Quinta-feira: Lógica Conversacional e Fluxos

7. Gerenciador de Conversas (services/conversation.py)

python

```
from typing import Dict, Optional, List
from datetime import datetime, timedelta
from sqlalchemy.orm import Session
from app.models.database import Conversation, Appointment, WaitingList, get_db
from app.services.whatsapp import WhatsAppService
from app.services.gestaods import GestaoDS
from app.utils.validators import ValidatorUtils
import logging
import re
```

```
logger = logging.getLogger(__name__)
```

```
class ConversationManager:
```

```
    def __init__(self):
```

```
        self.whatsapp = WhatsAppService()
```

```
        self.gestaods = GestaoDS()
```

```
        self.validator = ValidatorUtils()
```

```
    async def processar_mensagem(self, phone: str, message: str,
                                message_id: str, db: Session):
```

```
        """Processa mensagem e retorna resposta apropriada"""
```

```
        # Marcar mensagem como lida
```

```
        await self.whatsapp.mark_as_read(phone, message_id)
```

```
        # Buscar ou criar conversa
```

```
        conversa = self._get_or_create_conversation(phone, db)
```

```
        # Processar baseado no estado atual
```

```
        estado = conversa.state
```

```
        contexto = conversa.context or {}
```

```
        # Log para debug
```

```
        logger.info(f"Estado atual: {estado}, Mensagem: {message}")
```

```
        # Máquina de estados
```

```
        if estado == "inicio":
```

```
            await self._handle_inicio(phone, message, conversa, db)
```

```
        elif estado == "menu_principal":
```

```
            await self._handle_menu_principal(phone, message, conversa, db)
```

```
        elif estado == "aguardando_cpf":
```

```
            await self._handle_cpf(phone, message, conversa, db)
```

```
        elif estado == "escolhendo_data":
```

```
await self._handle_escolha_data(phone, message, conversa, db)
```

```
elif estado == "escolhendo_horario":
```

```
await self._handle_escolha_horario(phone, message, conversa, db)
```

```
elif estado == "confirmando_agendamento":
```

```
await self._handle_confirmacao(phone, message, conversa, db)
```

```
elif estado == "visualizando_agendamentos":
```

```
await self._handle_visualizar_agendamentos(phone, message, conversa, db)
```

```
elif estado == "cancelando_consulta":
```

```
await self._handle_cancelamento(phone, message, conversa, db)
```

```
elif estado == "lista_espera":
```

```
await self._handle_lista_espera(phone, message, conversa, db)
```

```
else:
```

```
# Estado desconhecido, reiniciar
```

```
conversa.state = "inicio"
```

```
db.commit()
```

```
await self._handle_inicio(phone, message, conversa, db)
```

```
async def _handle_inicio(self, phone: str, message: str,  
                        conversa: Conversation, db: Session):
```

```
    """Handler do estado inicial"""
```

```
# Enviar saudação e menu
```

```
saudacao = self._get_saudacao()
```

```
menu_text = f"""
```

```
{saudacao} Bem-vindo(a) à *{settings.clinic_name}*! 🏥
```

Sou seu assistente virtual e estou aqui para ajudar com seus agendamentos.

Como posso ajudar você hoje?

* **1** * - Agendar consulta

* **2** * - Ver meus agendamentos

* **3** * - Cancelar consulta

* **4** * - Lista de espera

* **5** * - Falar com atendente

Digite o número da opção desejada.

```
"""
```

```
await self.whatsapp.send_text(phone, menu_text)
```

```

# Atualizar estado
conversa.state = "menu_principal"
db.commit()

async def _handle_menu_principal(self, phone: str, message: str,
                                conversa: Conversation, db: Session):
    """Handler do menu principal"""

    opcao = message.strip()

    if opcao == "1":
        await self.whatsapp.send_text(
            phone,
            "Vamos agendar sua consulta! 📅\n\n"
            "Por favor, digite seu *CPF* (apenas números):"
        )
        conversa.state = "aguardando_cpf"
        conversa.context = {"acao": "lista_espera"}

    elif opcao == "5":
        await self.whatsapp.send_text(
            phone,
            "Vou transferir você para um atendente! 🧑‍💻\n\n"
            "Em breve alguém da nossa equipe entrará em contato.\n\n"
            "Horário de atendimento:\n"
            "📅 Segunda a Sexta: 8h às 18h\n"
            "📅 Sábado: 8h às 12h"
        )
        conversa.state = "inicio"
        # Aqui poderia notificar a equipe

    else:
        await self.whatsapp.send_text(
            phone,
            "Opção inválida! 😊\n\n"
            "Por favor, digite um número de *1 a 5*."
        )

    db.commit()

async def _handle_cpf(self, phone: str, message: str,
                     conversa: Conversation, db: Session):
    """Handler para validação de CPF"""

    # Limpar CPF
    cpf = re.sub(r'[^0-9]', '', message)

```

Validar CPF

```
if not self.validator.validar_cpf(cpf):  
    await self.whatsapp.send_text(  
        phone,  
        "❌ CPF inválido!\n\n"  
        "Por favor, digite um CPF válido (apenas números):"  
    )  
    return
```

Buscar paciente na API

```
paciente = await self.gestaods.buscar_paciente_cpf(cpf)  
  
if not paciente:  
    await self.whatsapp.send_text(  
        phone,  
        "❌ CPF não encontrado em nosso sistema.\n\n"  
        "Por favor, verifique o número e tente novamente.\n\n"  
        "Se você é um novo paciente, entre em contato "  
        "pelo telefone para realizar seu cadastro.\n\n"  
        "📞 (31) 9999-9999"  
    )  
    conversa.state = "inicio"  
    db.commit()  
    return
```

Salvar dados do paciente no contexto

```
contexto = conversa.context or {}  
contexto['paciente'] = {  
    'id': paciente.get('id'),  
    'nome': paciente.get('nome'),  
    'cpf': cpf,  
    'telefone': paciente.get('telefone', phone)  
}  
conversa.context = contexto
```

Continuar fluxo baseado na ação

```
acao = contexto.get('acao')  
  
if acao == "agendar":  
    await self._iniciar_agendamento(phone, paciente, conversa, db)  
elif acao == "visualizar":  
    await self._mostrar_agendamentos(phone, paciente, conversa, db)  
elif acao == "cancelar":  
    await self._iniciar_cancelamento(phone, paciente, conversa, db)  
elif acao == "lista_espera":  
    await self._adicionar_lista_espera(phone, paciente, conversa, db)
```

```
db.commit()
```

```
async def _iniciar_agendamento(self, phone: str, paciente: Dict,  
                                conversa: Conversation, db: Session):
```

```
    """Inicia processo de agendamento"""
```

```
    nome = paciente.get('nome', 'Paciente')
```

```
    # Gerar opções de datas (próximos 7 dias úteis)
```

```
    datas_disponiveis = self._gerar_datas_disponiveis()
```

```
    mensagem = f"""
```

```
Olá, *{nome}*! 😊
```

Vamos agendar sua consulta.

```
📅 *Escolha uma data:*
```

```
"""
```

```
    # Adicionar opções de data
```

```
    for i, data in enumerate(datas_disponiveis, 1):
```

```
        mensagem += f"\n*{i}* - {data['formatado']}"
```

```
    mensagem += "\n\nDigite o número da data desejada:"
```

```
    await self.whatsapp.send_text(phone, mensagem)
```

```
    # Salvar datas no contexto
```

```
    contexto = conversa.context
```

```
    contexto['datas_disponiveis'] = datas_disponiveis
```

```
    conversa.context = contexto
```

```
    conversa.state = "escolhendo_data"
```

```
async def _handle_escolha_data(self, phone: str, message: str,  
                                conversa: Conversation, db: Session):
```

```
    """Handler para escolha de data"""
```

```
    try:
```

```
        opcao = int(message.strip())
```

```
        contexto = conversa.context
```

```
        datas = contexto.get('datas_disponiveis', [])
```

```
        if 1 <= opcao <= len(datas):
```

```
            data_escolhida = datas[opcao - 1]
```

```
            contexto['data_escolhida'] = data_escolhida
```

```
    # Buscar horários disponíveis para a data
```


```
data_inicio = datetime.strptime(data_escolhida['data'], '%Y-%m-%d')
data_fim = data_inicio + timedelta(days=1)
```

```
horarios = await self.gestaods.listar_horarios_disponiveis(
    data_inicio, data_fim
)
```

```
if not horarios:
    await self.whatsapp.send_text(
        phone,
        "😞 Não há horários disponíveis para esta data.\n\n"
        "Por favor, escolha outra data:"
    )
    return
```

```
# Mostrar horários disponíveis
mensagem = f"""
```

```
 Data: {data_escolhida['formatado']}*
```

```
 *Horários disponíveis:*
```

```
*****
```

```
for i, horario in enumerate(horarios[:8], 1): # Limitar a 8 opções
    mensagem += f"\n*{i}* - {horario['hora']}"
```

```
mensagem += "\n\nDigite o número do horário desejado:"
```

```
await self.whatsapp.send_text(phone, mensagem)
```

```
contexto['horarios_disponiveis'] = horarios
conversa.context = contexto
conversa.state = "escolhendo_horario"
```

```
else:
    await self.whatsapp.send_text(
        phone,
        "❌ Opção inválida!\n\n"
        "Por favor, escolha um número válido."
    )
```

```
except ValueError:
    await self.whatsapp.send_text(
        phone,
        "❌ Por favor, digite apenas o número da opção desejada."
    )
```

```
db.commit()
```

```
async def _handle_escolha_horario(self, phone: str, message: str,
```

```
    conversa: Conversation, db: Session):
```

```
    """Handler para escolha de horário"""
```

```
    try:
```

```
        opcao = int(message.strip())
```

```
        contexto = conversa.context
```

```
        horarios = contexto.get('horarios_disponiveis', [])
```

```
        if 1 <= opcao <= len(horarios):
```

```
            horario_escolhido = horarios[opcao - 1]
```


```
            contexto['horario_escolhido'] = horario_escolhido
```

```
            # Mostrar resumo para confirmação
```

```
            paciente = contexto.get('paciente', {})
```

```
            data = contexto.get('data_escolhida', {})
```

```
            mensagem = f"""
```

```
             *Confirmar agendamento:*
```

```
             Paciente: {paciente.get('nome')}*  
             Data: {data.get('formatado')}*  
             Horário: {horario_escolhido.get('hora')}*  
             Profissional: {horario_escolhido.get('profissional')}*  
  
            *Confirma o agendamento?*
```

```
            *1* -  Sim, confirmar
```

```
            *2* -  Não, cancelar
```

```
            """
```

```
            await self.whatsapp.send_text(phone, mensagem)
```

```
            conversa.context = contexto
```

```
            conversa.state = "confirmando_agendamento"
```

```
        else:
```

```
            await self.whatsapp.send_text(
```

```
                phone,
```

```
                " Opção inválida!\n\n"
```

```
                "Por favor, escolha um número válido."
```

```
            )
```

```
    except ValueError:
```

```
        await self.whatsapp.send_text(
```

```
            phone,
```


"❌ Por favor, digite apenas o número da opção desejada."

)

db.commit()

```
async def _handle_confirmacao(self, phone: str, message: str,
```

```
    conversa: Conversation, db: Session):
```

```
    """Handler para confirmação de agendamento"""
```

```
    opcao = message.strip()
```

```
    if opcao == "1":
```

```
        contexto = conversa.context
```

```
        paciente = contexto.get('paciente', {})
```

```
        data = contexto.get('data_escolhida', {})
```

```
        horario = contexto.get('horario_escolhido', {})
```

```
        # Construir data/hora completa
```

```
        data_hora_str = f"{data['data']} {horario['hora']}"
```

```
        data_hora = datetime.strptime(data_hora_str, "%Y-%m-%d %H:%M")
```

```
        # Criar agendamento na API
```

```
        agendamento = await self.gestaods.criar_agendamento(
```

```
            paciente_id=paciente['id'],
```

```
            data_hora=data_hora,
```

```
            tipo="consulta",
```

```
            observacoes="Agendado via WhatsApp"
```

```
)
```

```
    if agendamento:
```

```
        # Salvar no banco local para lembretes
```

```
        novo_agendamento = Appointment(
```

```
            patient_id=str(paciente['id']),
```

```
            patient_name=paciente['nome'],
```

```
            patient_phone=phone,
```

```
            appointment_date=data_hora,
```

```
            appointment_type="consulta",
```

```
            status="scheduled"
```

```
)
```

```
        db.add(novo_agendamento)
```


```
        db.commit()
```


```
        # Enviar confirmação
```


```
        mensagem = f"""
```


✅ *Consulta agendada com sucesso!*

📋 *Detalhes do agendamento:*

 Paciente: {paciente.get('nome')}

 Data: {data.get('formatado')}

 Horário: {horario.get('hora')}

 Profissional: {horario.get('profissional')}

 *Endereço:*

Clínica Gabriela Nassif
Rua Example, 123 - Savassi
Belo Horizonte - MG

 *Lembretes:*

- Chegue com 15 minutos de antecedência
- Traga documento com foto
- Traga carteira do convênio (se aplicável)

Você receberá um lembrete 24h antes da consulta.

Obrigado por escolher nossa clínica! 😊

```
await self.whatsapp.send_text(phone, mensagem)
```

```
# Verificar se há alguém na lista de espera para notificar
```

```
await self._verificar_lista_espera_para_outras_datas(db)
```

```
else:
```

```
await self.whatsapp.send_text(
    phone,
    "❌ Erro ao agendar consulta.\n\n"
    "Por favor, tente novamente ou entre em contato."
)
```

```
# Resetar conversa
```

```
conversa.state = "inicio"
```

```
conversa.context = {}
```

```
elif opcao == "2":
```

```
await self.whatsapp.send_text(
    phone,
    "❌ Agendamento cancelado.\n\n"
    "Se desejar, podemos tentar outro horário.\n\n"
    "Digite *1* para voltar ao menu principal."
)
conversa.state = "inicio"
conversa.context = {}
```

```
else:
```

```
await self.whatsapp.send_text(
    phone,
    "Por favor, digite:\n"
    "*1* para confirmar\n"
    "*2* para cancelar"
)
```

```
db.commit()
```

```
async def _mostrar_agendamentos(self, phone: str, paciente: Dict,
                                conversa: Conversation, db: Session):
```

```
    """Mostra agendamentos do paciente"""
```

```
    agendamentos = await self.gestaods.listar_agendamentos_paciente(
        paciente['id']
    )
```

```
    if not agendamentos:
```

```
        await self.whatsapp.send_text(
            phone,
            "📅 Você não possui agendamentos futuros.\n\n"
            "Digite *1* para agendar uma consulta\n"
            "Digite *0* para voltar ao menu"
        )
```

```
    else:
```

```
        mensagem = f"📅 *Seus agendamentos:*\n\n"
```

```
        for i, ag in enumerate(agendamentos[:5], 1): # Limitar a 5
```

```
            data = datetime.fromisoformat(ag['data_hora'])
```

```
            mensagem += (
```

```
                f"*{i}*\n"
```

```
                f"📅 {data.strftime('%d/%m/%Y')}\n"
```

```
                f"🕒 {data.strftime('%H:%M')}\n"
```

```
                f"👤 {ag.get('profissional', 'Dr(a). Gabriela')}\n"
```

```
                f"📄 Status: {ag.get('status', 'Agendado')}\n\n"
```

```
            )
```

```
        mensagem += "Digite *0* para voltar ao menu"
```

```
        await self.whatsapp.send_text(phone, mensagem)
```

```
        conversa.state = "visualizando_agendamentos"
```

```
        db.commit()
```

```
def _get_or_create_conversation(self, phone: str, db: Session) -> Conversation:
```

```
    """Busca ou cria uma conversa"""
```

```
    conversa = db.query(Conversation).filter_by(phone=phone).first()
```

```
if not conversa:
    conversa = Conversation(phone=phone)
    db.add(conversa)
    db.commit()
```

```
return conversa
```

```
def _get_saudacao(self) -> str:
    """Retorna saudação baseada no horário"""
    hora = datetime.now().hour
```

```
if hora < 12:
    return "🌄 Bom dia!"
elif hora < 18:
    return "🌞 Boa tarde!"
else:
    return "🌙 Boa noite!"
```

```
def _gerar_datas_disponiveis(self, dias: int = 7) -> List[Dict]:
    """Gera lista de datas disponíveis (dias úteis)"""
    datas = []
    data_atual = datetime.now()

    while len(datas) < dias:
        data_atual += timedelta(days=1)

        # Pular fins de semana
        if data_atual.weekday() < 5: # 0-4 = Seg-Sex
            datas.append({
                'data': data_atual.strftime('%Y-%m-%d'),
                'formatado': data_atual.strftime('%d/%m/%Y - %A').replace(
                    'Monday', 'Segunda').replace(
                    'Tuesday', 'Terça').replace(
                    'Wednesday', 'Quarta').replace(
                    'Thursday', 'Quinta').replace(
                    'Friday', 'Sexta')
            })

    return datas
```

```
async def _verificar_lista_espera_para_outras_datas(self, db: Session):
    """Verifica se há pessoas na lista de espera para notificar"""
    # Implementação futura
    pass
```

8. Utilitários de Validação (utils/validators.py)

python

```
import re
from datetime import datetime
from typing import Optional
```

```
class ValidatorUtils:
```

```
    @staticmethod
```

```
    def validar_cpf(cpf: str) -> bool:
```

```
        """Valida CPF brasileiro"""
```

```
        # Remove caracteres não numéricos
```

```
        cpf = re.sub(r'^0-9', '', cpf)
```

```
        # Verifica se tem 11 dígitos
```

```
        if len(cpf) != 11:
```

```
            return False
```

```
        # Verifica se todos os dígitos são iguais
```

```
        if cpf == cpf[0] * 11:
```

```
            return False
```

```
        # Validação dos dígitos verificadores
```

```
        def calcular_digito(cpf_parcial):
```

```
            soma = 0
```

```
            for i, digito in enumerate(cpf_parcial):
```

```
                soma += int(digito) * (len(cpf_parcial) + 1 - i)
```

```
            resto = soma % 11
```

```
            return '0' if resto < 2 else str(11 - resto)
```

```
        # Primeiro dígito
```

```
        if calcular_digito(cpf[:9]) != cpf[9]:
```

```
            return False
```

```
        # Segundo dígito
```

```
        if calcular_digito(cpf[:10]) != cpf[10]:
```

```
            return False
```

```
        return True
```

```
    @staticmethod
```

```
    def validar_telefone(telefone: str) -> bool:
```

```
        """Valida número de telefone brasileiro"""
```

```
        # Remove caracteres não numéricos
```

```
        telefone = re.sub(r'^0-9', '', telefone)
```

```
        # Verifica comprimento (com ou sem código do país)
```

```
        if len(telefone) == 11: # Sem código do país
```

```

return telefone[2] == '9'
elif len(telefone) == 13: # Com código do país (55)
    return telefone[:2] == '55' and telefone[4] == '9'

return False

```

@staticmethod

```

def validar_data(data_str: str, formato: str = "%d/%m/%Y") -> Optional[datetime]:
    """Valida e converte string para data"""
    try:
        return datetime.strptime(data_str, formato)
    except ValueError:
        return None

```

@staticmethod

```

def validar_horario(horario_str: str) -> Optional[str]:
    """Valida formato de horário HH:MM"""
    pattern = r'^([0-1]?[0-9])2[0-3]:[0-5][0-9]'
    conversa.context = {"acao": "agendar"}

    elif opcao == "2":
        await self.whatsapp.send_text(
            phone,
            "Para ver seus agendamentos, preciso do seu *CPF*.\n\n"
            "Digite seu CPF (apenas números):"
        )
        conversa.state = "aguardando_cpf"
        conversa.context = {"acao": "visualizar"}

    elif opcao == "3":
        await self.whatsapp.send_text(
            phone,
            "Para cancelar uma consulta, preciso do seu *CPF*.\n\n"
            "Digite seu CPF (apenas números):"
        )
        conversa.state = "aguardando_cpf"
        conversa.context = {"acao": "cancelar"}

    elif opcao == "4":
        await self.whatsapp.send_text(
            phone,
            "Vou adicionar você na lista de espera! 📌\n\n"
            "Digite seu *CPF* (apenas números):"
        )
        conversa.state = "aguardando_cpf"
    if re.match(pattern, horario_str):
        return horario_str

```

```
return None
```

```
@staticmethod
```

```
def formatar_telefone(telefone: str) -> str:
```

```
    """Formata número de telefone para exibição"""
```

```
    telefone = re.sub(r'^0-9', '', telefone)
```

```
    if len(telefone) == 11:
```

```
        return f"({telefone[:2]}) {telefone[2:7]}-{telefone[7:]}"
```

```
    elif len(telefone) == 13:
```

```
        return f"+{telefone[:2]} ({telefone[2:4]}) {telefone[4:9]}-{telefone[9:]}"
```

```
    return telefone
```

```
@staticmethod
```

```
def formatar_cpf(cpf: str) -> str:
```

```
    """Formata CPF para exibição"""
```

```
    cpf = re.sub(r'^0-9', '', cpf)
```

```
    if len(cpf) == 11:
```

```
        return f"{cpf[:3]}.{cpf[3:6]}.{cpf[6:9]}-{cpf[9:]}"
```

```
    return cpf
```

9. Handlers de Webhook (handlers/webhook.py)

```
python
```



```
from fastapi import APIRouter, Request, HTTPException, Depends
from sqlalchemy.orm import Session
from app.models.database import get_db
from app.services.conversation import ConversationManager
import logging
```

```
logger = logging.getLogger(__name__)
router = APIRouter()
```

```
conversation_manager = ConversationManager()
```

```
@router.post("/webhook/message")
```

```
async def webhook_message(request: Request, db: Session = Depends(get_db)):
```

```
    """Recebe mensagens do WhatsApp via Z-API"""
```

```
    try:
```

```
        data = await request.json()
```

```
        # Log para debug
```

```
        logger.info(f"Webhook recebido: {data}")
```

```
        # Extrair dados da mensagem
```

```
        if data.get("type") == "ReceivedCallback":
```

```
            phone = data.get("phone", "").replace("@c.us", "")
```

```
            # Verificar se é mensagem de texto
```

```
            text_data = data.get("text", {})
```

```
            if text_data and "message" in text_data:
```

```
                message = text_data["message"]
```

```
                message_id = data.get("messageId", "")
```

```
            # Processar mensagem
```

```
            await conversation_manager.processar_mensagem(
                phone, message, message_id, db
            )
```

```
            return {"status": "success"}
```

```
        return {"status": "ignored", "reason": "not_text_message"}
```

```
    except Exception as e:
```

```
        logger.error(f"Erro no webhook: {str(e)}")
```

```
        raise HTTPException(status_code=500, detail=str(e))
```

```
@router.post("/webhook/status")
```

```
async def webhook_status(request: Request):
```

```
    """Recebe atualizações de status das mensagens"""
```

```
try:
    data = await request.json()

    # Log status para monitoramento
    logger.info(f"Status update: {data}")

    return {"status": "success"}

except Exception as e:
    logger.error(f"Erro no webhook status: {str(e)}")
    return {"status": "error"}

@router.post("/webhook/connected")
async def webhook_connected(request: Request):
    """Notificação quando WhatsApp conecta/desconecta"""
    try:
        data = await request.json()

        connected = data.get("connected", False)

        if connected:
            logger.info("WhatsApp conectado com sucesso!")
        else:
            logger.warning("WhatsApp desconectado!")

        return {"status": "success"}

    except Exception as e:
        logger.error(f"Erro no webhook connected: {str(e)}")
        return {"status": "error"}
```

10. Sistema de Lembretes (tasks/reminders.py)

```
python
```

```

from datetime import datetime, timedelta
from sqlalchemy.orm import Session
from app.models.database import Appointment, WaitingList, SessionLocal
from app.services.whatsapp import WhatsAppService
from app.services.gestaods import GestaoDS
import logging

logger = logging.getLogger(__name__)

class ReminderService:
    def __init__(self):
        self.whatsapp = WhatsAppService()
        self.gestaods = GestaoDS()

    async def enviar_lembretes_diarios(self):
        """Envia lembretes para consultas do dia seguinte"""
        try:
            db = SessionLocal()

            # Buscar consultas de amanhã
            amanha = datetime.now() + timedelta(days=1)
            inicio_dia = amanha.replace(hour=0, minute=0, second=0)
            fim_dia = amanha.replace(hour=23, minute=59, second=59)

            consultas = db.query(Appointment).filter(
                Appointment.appointment_date >= inicio_dia,
                Appointment.appointment_date <= fim_dia,
                Appointment.status == "scheduled",
                Appointment.reminder_sent == False
            ).all()

            logger.info(f"Enviando {len(consultas)} lembretes")

            for consulta in consultas:
                await self._enviar_lembrete_individual(consulta, db)


            db.close()

        except Exception as e:
            logger.error(f"Erro ao enviar lembretes: {str(e)}")

    async def _enviar_lembrete_individual(self, consulta: Appointment, db: Session):
        """Envia lembrete individual"""
        try:
            data_hora = consulta.appointment_date

```

```
mensagem = f''''
```

```
 *Lembrete de Consulta*
```

```
Olá, {consulta.patient_name}!
```

Este é um lembrete da sua consulta amanhã:

```
 Data: {data_hora.strftime('%d/%m/%Y')}
```

```
 Horário: {data_hora.strftime('%H:%M')}
```

```
 Local: Clínica Gabriela Nassif
```

Por favor, confirme sua presença:

1 -  Confirmar presença

2 -  Não poderei comparecer

3 -  Reagendar

Digite a opção desejada.

```
''''
```

```
# Enviar lembrete
```

```
await self.whatsapp.send_text(consulta.patient_phone, mensagem)
```

```
# Marcar como enviado
```

```
consulta.reminder_sent = True
```

```
db.commit()
```

```
logger.info(f"Lembrete enviado para {consulta.patient_name}")
```

```
except Exception as e:
```

```
logger.error(f"Erro ao enviar lembrete: {str(e)}")
```

```
async def verificar_cancelamentos(self):
```

```
    """Verifica cancelamentos e notifica lista de espera"""
```

```
    try:
```

```
        db = SessionLocal()
```

```
# Buscar consultas canceladas recentemente
```

```
uma_hora_atras = datetime.now() - timedelta(hours=1)
```

```
consultas_canceladas = db.query(Appointment).filter(
```

```
    Appointment.status == "cancelled",
```

```
    Appointment.updated_at >= uma_hora_atras
```

```
).all()
```

```
for consulta in consultas_canceladas:
```

```
    await self._notificar_lista_espera(consulta, db)
```

```
db.close()
```

```
except Exception as e:
```

```
    logger.error(f"Erro ao verificar cancelamentos: {str(e)}")
```

```
async def _notificar_lista_espera(self, consulta_cancelada: Appointment, db: Session):
```

```
    """Notifica pessoas na lista de espera sobre vaga disponível"""
```

```
    try:
```

```
        # Buscar pessoas na lista de espera
```

```
        lista_espera = db.query(WaitingList).filter(
```

```
            WaitingList.notified == False
```

```
        ).order_by(WaitingList.priority.desc(), WaitingList.created_at).limit(3).all()
```

```
        data_hora = consulta_cancelada.appointment_date
```

```
        for pessoa in lista_espera:
```

```
            mensagem = f"""
```

```
🌈 *Vaga Disponível!*
```

```
Olá, {pessoa.patient_name}!
```

```
Surgiu uma vaga para consulta:
```

```
📅 Data: {data_hora.strftime("%d/%m/%Y")}
```

```
🕒 Horário: {data_hora.strftime("%H:%M")}
```

```
*Deseja agendar?*
```

```
*1* - ✅ Sim, quero a vaga!
```

```
*2* - ❌ Não posso neste horário
```

```
⚡ Responda rápido! Esta vaga pode ser preenchida por outra pessoa.
```

```
"""
```

```
        await self.whatsapp.send_text(pessoa.patient_phone, mensagem)
```

```
        # Marcar como notificado
```

```
        pessoa.notified = True
```

```
        db.commit()
```

```
        logger.info(f"Lista de espera notificada: {pessoa.patient_name}")
```

```
except Exception as e:
```

```
    logger.error(f"Erro ao notificar lista de espera: {str(e)}")
```

Dia 3 - Sexta-feira: Finalização e Deploy

11. Arquivo Principal (main.py)

python

```
from fastapi import FastAPI, Request
from fastapi.middleware.cors import CORSMiddleware
from apscheduler.schedulers.asyncio import AsyncIOScheduler
from contextlib import asynccontextmanager
import logging
import sys

from app.config import settings
from app.handlers.webhook import router as webhook_router
from app.tasks.reminders import ReminderService

# Configurar logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('chatbot.log'),
        logging.StreamHandler(sys.stdout)
    ]
)

logger = logging.getLogger(__name__)

# Scheduler global
scheduler = AsyncIOScheduler()
reminder_service = ReminderService()

@asynccontextmanager
async def lifespan(app: FastAPI):
    """Gerencia ciclo de vida da aplicação"""
    # Startup
    logger.info("Iniciando aplicação...")

    # Configurar tarefas agendadas
    scheduler.add_job(
        reminder_service.enviar_lembretes_diarios,
        'cron',
        hour=settings.reminder_hour,
        minute=settings.reminder_minute
    )

    scheduler.add_job(
        reminder_service.verificar_cancelamentos,
        'interval',
        minutes=30
    )
```

```
scheduler.start()
logger.info("Scheduler iniciado")

yield

# Shutdown
scheduler.shutdown()
logger.info("Aplicação encerrada")

# Criar aplicação FastAPI
app = FastAPI(
    title="Chatbot Clínica WhatsApp",
    description="Assistente virtual para agendamento de consultas",
    version="1.0.0",
    lifespan=lifespan
)

# Configurar CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Incluir routers
app.include_router(webhook_router, prefix="/webhook", tags=["webhook"])

@app.get("/")
async def root():
    """Endpoint de saúde"""
    return {
        "status": "online",
        "service": "Chatbot Clínica",
        "version": "1.0.0"
    }

@app.get("/health")
async def health_check():
    """Verificação de saúde detalhada"""
    return {
        "status": "healthy",
        "database": "connected",
        "whatsapp": "connected",
        "scheduler": scheduler.running
    }
```



```
}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(
        "app.main:app",
        host=settings.app_host,
        port=settings.app_port,
        reload=settings.debug
    )

    conversa.context = {"acao": "agendar"}

    elif opcao == "2":
        await self.whatsapp.send_text(
            phone,
            "Para ver seus agendamentos, preciso do seu *CPF*.\n\n"
            "Digite seu CPF (apenas números):"
        )
        conversa.state = "aguardando_cpf"
        conversa.context = {"acao": "visualizar"}

    elif opcao == "3":
        await self.whatsapp.send_text(
            phone,
            "Para cancelar uma consulta, preciso do seu *CPF*.\n\n"
            "Digite seu CPF (apenas números):"
        )
        conversa.state = "aguardando_cpf"
        conversa.context = {"acao": "cancelar"}

    elif opcao == "4":
        await self.whatsapp.send_text(
            phone,
            "Vou adicionar você na lista de espera! 📅\n\n"
            "Digite seu *CPF* (apenas números):"
        )
        conversa.state = "aguardando_cpf"
```