■ Proyecto Flask - Código Python (Parte 2)

Total de archivos: 6

Archivo 1: ./models/monitor.py

```
import asyncio■
from datetime import datetime■
from services.maas_client import obtener_estado_actual■
from services.telegram_service import enviar_notificacion_telegram■
■
class MonitorMaquinas:■
    def __init__(self):■
        self.estados_anteriores = {}■
        self.monitoreo_activo = False■
        self.intervalo = 30■
    ■
    def detectar_cambios(self, estados_actuales):■
        cambios = []■
        ■
        for hostname, estado_actual in estados_actuales.items():■
            estado_anterior = self.estados_anteriores.get(hostname, {})■
            ■
            if hostname not in self.estados_anteriores:■
                cambios.append(f"■ <b>Nueva máquina detectada:</b> {hostname} ({estado_actual['ip']}) - Estado: {estac
            else:■
                if estado_anterior.get('power_state') != estado_actual['power_state']:■
                    if estado_actual['power_state'] == 'on':■
                        cambios.append(f"■ <b>Máquina encendida:</b> {hostname} ({estado_actual['ip']})")■
                    elif estado_actual['power_state'] == 'off':■
                        cambios.append(f"■ <b>Máquina apagada:</b> {hostname} ({estado_actual['ip']})")■
                    else:■
                        cambios.append(f"■ <b>Estado cambiado:</b> {hostname} ({estado_actual['ip']}) - Nuevo estado:
        ■
        for hostname in self.estados_anteriores:■
            if hostname not in estados_actuales:■
                cambios.append(f"■ <b>Máquina desaparecida:</b> {hostname}")■
        ■
        return cambios■
    ■
    async def verificar_estados(self):■
        try:■
            estados_actuales = await obtener_estado_actual()■
            ■
            if self.estados_anteriores:■
                cambios = self.detectar_cambios(estados_actuales)■
                ■
                for cambio in cambios:■
                    mensaje_completo = f"■ <b>Notificación MAAS</b>\n{cambio}\n■ {datetime.now().strftime('%Y-%m-%d %H
                    if enviar_notificacion_telegram(mensaje_completo):■
                        print(f"Notificación enviada: {cambio}")■
                    else:■
                        print(f"Error enviando notificación: {cambio}")■
                    await asyncio.sleep(1)■
            ■
            self.estados_anteriores = estados_actuales■
            ■
        except Exception as e:■
            print(f"Error en verificación de estados: {e}")■
    ■
    async def iniciar_monitoreo(self):■
        self.monitoreo_activo = True■
        print("■ Iniciando monitoreo de máquinas MAAS...")■
        ■
        try:■
            self.estados_anteriores = await obtener_estado_actual()■
            print(f"■ Estado inicial capturado: {len(self.estados_anteriores)} máquinas")■
        except Exception as e:■
            print(f"Error en verificación inicial: {e}")■
        ■
        while self.monitoreo_activo:■
            try:■
                await self.verificar_estados()■
                await asyncio.sleep(self.intervalo)■
            except Exception as e:■
                print(f"Error en bucle de monitoreo: {e}")■
                await asyncio.sleep(self.intervalo)■
    ■
    def detener_monitoreo(self):■
        self.monitoreo_activo = False■
        print("■■ Monitoreo detenido")
```

Archivo 2: ./utils/helpers.py

```
def serializar_objeto_simple(obj):■
    if obj is None:■
        return None■
    elif isinstance(obj, (str, int, float, bool)):■
        return obj■
    elif isinstance(obj, list):■
        return [serializar_objeto_simple(item) for item in obj]■
    elif isinstance(obj, dict):■
        return {key: serializar_objeto_simple(value) for key, value in obj.items()}■
    elif hasattr(obj, '__dict__'):■
        result = {}■
        for key, value in obj.__dict__.items():■
            if not key.startswith('_'):■
                try:■
                    result[key] = serializar_objeto_simple(value)■
                except:■
                    result[key] = str(value)■
        return result■
    else:■
        return str(obj)
```

Archivo 3: ./services/chat_service.py

```
import asyncio■
import re■
from config import CORTESIAS■
from services.maas_client import listar_maquinas, listar_subredes, encender_maquina, apagar_maquina, buscar_maquina_po
from services.gemini_service import generar_respuesta_gemini■
■
async def responder_pregunta(pregunta):■
    pregunta_lower = pregunta.lower()■
■
    # Filtrar cortesías■
    if any(c in pregunta_lower for c in CORTESIAS):■
        return "¡De nada! ■"■
■
    # DETECTAR COMANDOS DE CONTROL DE MÁQUINAS■
    # Comando para encender máquina■
    if any(palabra in pregunta_lower for palabra in ["enciende", "encienda", "prende", "prenda", "power on", "encender
        # Extraer el identificador de la máquina■
        identificador = None■
        ■
        # Buscar por nombre de máquina■
        maquinas_texto = await listar_maquinas()■
        for linea in maquinas_texto.split('\n'):■
            if 'MÁQUINA:' in linea:■
                partes = linea.split('(')■
                if len(partes) > 1:■
                    nombre_maquina = partes[0].replace('■ MÁQUINA:', '').strip()■
                    system_id = partes[1].replace(')', '').strip()■
                    ■
                    if nombre_maquina.lower() in pregunta_lower:■
                        identificador = nombre_maquina■
                        break■
                    elif system_id.lower() in pregunta_lower:■
                        identificador = system_id■
                        break■
        ■
        # Buscar por IP■
        if not identificador:■
            ip_pattern = r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b'■
            ips = re.findall(ip_pattern, pregunta)■
            if ips:■
                maquina_por_ip = await buscar_maquina_por_ip(ips[0])■
                if maquina_por_ip:■
                    identificador = maquina_por_ip.hostname■
        ■
        if not identificador:■
            # Si no se encontró identificador, pedir clarificación■
            return "■ ¿Qué máquina quieres encender? Por favor, especifica el nombre o ID de la máquina."■
        ■
        # Ejecutar comando de encender■
        resultado = await encender_maquina(identificador)■
        return resultado■
■
    # Comando para apagar máquina■
    elif any(palabra in pregunta_lower for palabra in ["apaga", "apague", "apagar", "power off", "apagado"]):■
        # Extraer el identificador de la máquina■
        identificador = None■
        ■
        # Buscar por nombre de máquina■
        maquinas_texto = await listar_maquinas()■
        for linea in maquinas_texto.split('\n'):■
            if 'MÁQUINA:' in linea:■
                partes = linea.split('(')■
                if len(partes) > 1:■
                    nombre_maquina = partes[0].replace('■ MÁQUINA:', '').strip()■
                    system_id = partes[1].replace(')', '').strip()■
                    ■
                    if nombre_maquina.lower() in pregunta_lower:■
                        identificador = nombre_maquina■
                        break■
                    elif system_id.lower() in pregunta_lower:■
                        identificador = system_id■
                        break■
        ■
        # Buscar por IP■
        if not identificador:■
```

```
            ip_pattern = r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b'■
            ips = re.findall(ip_pattern, pregunta)■
            if ips:■
                maquina_por_ip = await buscar_maquina_por_ip(ips[0])■
                if maquina_por_ip:■
                    identificador = maquina_por_ip.hostname■
        ■
        if not identificador:■
            # Si no se encontró identificador, pedir clarificación■
            return "■ ¿Qué máquina quieres apagar? Por favor, especifica el nombre o ID de la máquina."■
        ■
        # Ejecutar comando de apagar■
        resultado = await apagar_maquina(identificador)■
        return resultado■
■
    # Subredes■
    if "subred" in pregunta_lower:■
        subredes_texto = await listar_subredes()■
        prompt = f"""■
INFORMACIÓN DE SUBREDES EN MAAS:■
{subredes_texto}■
■
PREGUNTA DEL USUARIO: {pregunta}■
■
Responde en español de forma clara y amigable, usando exactamente la información proporcionada.■
"""■
        return generar_respuesta_gemini(prompt)■
■
    # Máquinas (consultas informativas)■
    maquinas_texto = await listar_maquinas()■
    ■
    # DETECTAR TIPO DE PREGUNTA PARA ADAPTAR LA RESPUESTA■
    if any(palabra in pregunta_lower for palabra in ["ram", "memoria"]):■
        prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre la memoria RAM■
- No menciones información sobre almacenamiento, CPUs, estado de encendido, etc.■
- Sé conciso y directo■
- Usa los valores EXACTOS de la información proporcionada■
■
Responde en español:■
"""■
    elif any(palabra in pregunta_lower for palabra in ["almacenamiento", "disco", "disco duro", "storage", "gb", "tera
        prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre el almacenamiento■
- No menciones información sobre RAM, CPUs, estado de encendido, etc.■
- Sé conciso y directo■
- Usa los valores EXACTOS de la información proporcionada■
■
Responde en español:■
"""■
    elif any(palabra in pregunta_lower for palabra in ["cpu", "procesador", "núcleo", "nucleo", "procesadores"]):■
        prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre los CPUs/procesadores■
- No menciones información sobre RAM, almacenamiento, estado de encendido, etc.■
- Sé conciso y directo■
- Usa los valores EXACTOS de la información proporcionada■
■
Responde en español:■
"""■
```

```
    elif any(palabra in pregunta_lower for palabra in ["encend", "apag", "power", "on", "off", "estado"]):■
        prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre el estado de encendido/apagado■
- No menciones información sobre RAM, almacenamiento, CPUs, etc.■
- Usa los términos EXACTOS: ■ ENCENDIDA, ■ APAGADA, ■ DESCONOCIDO■
- Sé conciso y directo■
■
Responde en español:■
"""■
    elif any(palabra in pregunta_lower for palabra in ["ip", "dirección", "direccion", "red", "network"]):■
        prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre las direcciones IP■
- No menciones información sobre RAM, almacenamiento, CPUs, estado de encendido, etc.■
- Sé conciso y directo■
- Usa los valores EXACTOS de la información proporcionada■
■
Responde en español:■
"""■
    elif any(palabra in pregunta_lower for palabra in ["información", "info", "detalles", "resumen", "todo", "general"
        prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Proporciona un resumen COMPLETO pero BIEN ESTRUCTURADO de todas las máquinas■
- Para CADA máquina, incluye: estado de encendido, IP, RAM, almacenamiento, CPUs y SO■
- Usa un formato CLARO y ORGANIZADO■
- Separa cada máquina con una línea en blanco■
- Mantén la información CONCISA pero COMPLETA■
- Usa los valores EXACTOS de la información proporcionada■
- Incluye los emojis para hacerlo más visual■
■
EJEMPLO DE FORMATO CORRECTO:■
"■ MÁQUINA: maquinaprueba (7mdht4)■
■ Estado: ■ ENCENDIDA | ■ IP: 172.16.25.201■
■ RAM: 2 GB | ■ Almacenamiento: 21.0 GB | ■ CPUs: 1 núcleo■
■ SO: ubuntu jammy"■
■
Responde en español:■
"""■
    else:■
        prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Analiza qué información es RELEVANTE para responder esta pregunta específica■
- Responde de forma CONCISA, mencionando solo la información necesaria■
- Si la pregunta es sobre un aspecto concreto, habla solo de ese aspecto■
- Si es una pregunta general, da un resumen breve pero completo■
- Usa los valores EXACTOS de la información proporcionada■
- No des información innecesaria o redundante■
■
Responde en español:■
"""■
■
    return generar_respuesta_gemini(prompt)
```

Archivo 4: ./services/maas_client.py

```
import asyncio■
from datetime import datetime■
from maas.client import connect■
import re■
import threading■
■
from config import MAAS_URL, MAAS_API_KEY■
from services.telegram_service import enviar_notificacion_telegram■
from utils.helpers import serializar_objeto_simple■
■
# ==========================■
# Funciones básicas de MAAS■
# ==========================■
■
async def obtener_maquinas():■
    """Obtiene lista de todas las máquinas"""■
    client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
    return await client.machines.list()■
■
async def obtener_estado_actual():■
    """Obtiene el estado actual de todas las máquinas"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        machines = await client.machines.list()■
        estados_actuales = {}■
        ■
        for m in machines:■
            m_full = await client.machines.get(m.system_id)■
            power_state = m_full._data.get("power_state", "unknown")■
            estados_actuales[m_full.hostname] = {■
                "power_state": power_state,■
                "system_id": m_full.system_id,■
                "ip": m_full.ip_addresses[0] if m_full.ip_addresses else "Sin IP"■
            }■
        ■
        return estados_actuales■
    except Exception as e:■
        print(f"Error obteniendo estados: {e}")■
        return {}■
■
async def listar_maquinas():■
    """Lista todas las máquinas en formato texto legible"""■
    client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
    machines = await client.machines.list()■
    lista_texto = ""■
■
    for m in machines:■
        m_full = await client.machines.get(m.system_id)■
■
        # INFORMACIÓN BÁSICA■
        hostname = m_full.hostname■
        system_id = m_full.system_id■
        status_name = m_full.status_name■
        ■
        # ESTADO DE ENCENDIDO■
        power_state = m_full._data.get("power_state", "unknown")■
        if power_state == "on":■
            encendido = "■ ENCENDIDA"■
        elif power_state == "off":■
            encendido = "■ APAGADA"■
        else:■
            encendido = "■ DESCONOCIDO"■
        ■
        # IP■
        ip_principal = m_full.ip_addresses[0] if m_full.ip_addresses else "Sin IP"■
        ■
        # HARDWARE■
        memoria_mb = m_full._data.get("memory", 0)■
        ram_gb = round(memoria_mb / 1024) if memoria_mb and memoria_mb > 0 else "Desconocida"■
        ■
        storage_mb = m_full._data.get("storage", 0)■
        storage_gb = round(storage_mb / 1024, 1) if storage_mb and storage_mb > 0 else "Desconocido"■
        ■
        cpu_count = m_full._data.get("cpu_count", "Desconocido")■
        ■
```

```
        # SISTEMA OPERATIVO■
        osystem = m_full.osystem■
        distro_series = m_full.distro_series■
        ■
        # ZONA Y POOL■
        zone_name = m_full.zone.name if m_full.zone else "default"■
        pool_name = m_full.pool.name if m_full.pool else "default"■
■
        lista_texto += (■
            f"■ MÁQUINA: {hostname} ({system_id})\n"■
            f"■ Estado MAAS: {status_name}\n"■
            f"■ Estado: {encendido}\n"■
            f"■ IP: {ip_principal}\n"■
            f"■ RAM: {ram_gb} GB\n"■
            f"■ Almacenamiento: {storage_gb} GB\n"■
            f"■ CPUs: {cpu_count} núcleos\n"■
            f"■ SO: {osystem} {distro_series}\n"■
            f"■■ Zona: {zone_name} | Pool: {pool_name}\n\n"■
            f"■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■\n\n"■
        )■
■
    return lista_texto■
■
async def obtener_subredes():■
    """Obtiene lista de todas las subredes"""■
    client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
    return await client.subnets.list()■
■
async def listar_subredes():■
    """Lista todas las subredes en formato texto legible"""■
    subnets = await obtener_subredes()■
    lista_texto = ""■
    for s in subnets:■
        cidr = getattr(s, "cidr", "Desconocido")■
        name = getattr(s, "name", "Sin nombre")■
        vlan = getattr(s, "vlan", "No asignada")■
        lista_texto += f"- Subred: {name}, CIDR: {cidr}, VLAN: {vlan}\n"■
    return lista_texto■
■
# ==========================■
# Control de máquinas■
# ==========================■
■
async def encender_maquina(identificador):■
    """Enciende una máquina por hostname o system_id"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquinas = await client.machines.list()■
        ■
        maquina_encontrada = None■
        for m in maquinas:■
            m_full = await client.machines.get(m.system_id)■
            if (m_full.hostname.lower() == identificador.lower() or ■
                m_full.system_id.lower() == identificador.lower()):■
                maquina_encontrada = m_full■
                break■
        ■
        if not maquina_encontrada:■
            mensaje_error = f"■ <b>Error en comando</b>\n■ <b>Máquina no encontrada:</b> {identificador}\n■ {datetime
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_error)).start()■
            return f"■ No se encontró la máquina: {identificador}"■
        ■
        # Verificar estado actual■
        power_state = maquina_encontrada._data.get("power_state", "unknown")■
        if power_state == "on":■
            return f"■■ La máquina {maquina_encontrada.hostname} ya está encendida"■
        ■
        # Notificación de inicio de comando■
        ip_maquina = maquina_encontrada.ip_addresses[0] if maquina_encontrada.ip_addresses else "Sin IP"■
        mensaje_inicio = f"■ <b>Comando ejecutado</b>\n■ <b>Encendiendo:</b> {maquina_encontrada.hostname} ({ip_maqui
        threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_inicio)).start()■
        ■
        # Encender la máquina■
        await maquina_encontrada.power_on()■
        await asyncio.sleep(5)■
        ■
        # Verificar nuevo estado■
```

```
        maquina_actualizada = await client.machines.get(maquina_encontrada.system_id)■
        nuevo_estado = maquina_actualizada._data.get("power_state", "unknown")■
        ■
        if nuevo_estado == "on":■
            mensaje_exito = f"■ <b>Comando completado</b>\n■ <b>Máquina encendida:</b> {maquina_encontrada.hostname}
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_exito)).start()■
            return f"■ Máquina {maquina_encontrada.hostname} encendida exitosamente"■
        else:■
            return f"■■ La máquina {maquina_encontrada.hostname} se está encendiendo (puede tardar unos momentos)"■
            ■
    except Exception as e:■
        mensaje_error = f"■ <b>Error en comando</b>\n■ <b>Error al encender:</b> {identificador}\n■ {str(e)}\n■ {date
        threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_error)).start()■
        return f"■ Error al encender la máquina: {e}"■
■
async def apagar_maquina(identificador):■
    """Apaga una máquina por hostname o system_id"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquinas = await client.machines.list()■
        ■
        maquina_encontrada = None■
        for m in maquinas:■
            m_full = await client.machines.get(m.system_id)■
            if (m_full.hostname.lower() == identificador.lower() or ■
                m_full.system_id.lower() == identificador.lower()):■
                maquina_encontrada = m_full■
                break■
        ■
        if not maquina_encontrada:■
            mensaje_error = f"■ <b>Error en comando</b>\n■ <b>Máquina no encontrada:</b> {identificador}\n■ {datetime
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_error)).start()■
            return f"■ No se encontró la máquina: {identificador}"■
        ■
        # Verificar estado actual■
        power_state = maquina_encontrada._data.get("power_state", "unknown")■
        if power_state == "off":■
            return f"■■ La máquina {maquina_encontrada.hostname} ya está apagada"■
        ■
        # Notificación de inicio de comando■
        ip_maquina = maquina_encontrada.ip_addresses[0] if maquina_encontrada.ip_addresses else "Sin IP"■
        mensaje_inicio = f"■ <b>Comando ejecutado</b>\n■ <b>Apagando:</b> {maquina_encontrada.hostname} ({ip_maquina}
        threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_inicio)).start()■
        ■
        # Apagar la máquina■
        await maquina_encontrada.power_off()■
        await asyncio.sleep(5)■
        ■
        # Verificar nuevo estado■
        maquina_actualizada = await client.machines.get(maquina_encontrada.system_id)■
        nuevo_estado = maquina_actualizada._data.get("power_state", "unknown")■
        ■
        if nuevo_estado == "off":■
            mensaje_exito = f"■ <b>Comando completado</b>\n■ <b>Máquina apagada:</b> {maquina_encontrada.hostname} ({
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_exito)).start()■
            return f"■ Máquina {maquina_encontrada.hostname} apagada exitosamente"■
        else:■
            return f"■■ La máquina {maquina_encontrada.hostname} se está apagando (puede tardar unos momentos)"■
            ■
    except Exception as e:■
        mensaje_error = f"■ <b>Error en comando</b>\n■ <b>Error al apagar:</b> {identificador}\n■ {str(e)}\n■ {dateti
        threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_error)).start()■
        return f"■ Error al apagar la máquina: {e}"■
■
async def buscar_maquina_por_ip(ip):■
    """Busca una máquina por dirección IP"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquinas = await client.machines.list()■
        ■
        for m in maquinas:■
            m_full = await client.machines.get(m.system_id)■
            if m_full.ip_addresses and ip in m_full.ip_addresses:■
                return m_full■
        return None■
    except Exception as e:■
        print(f"Error buscando máquina por IP: {e}")■
```

```
        return None
async def buscar_maquina_por_nombre_o_id(identificador):
    """Busca una máquina por nombre o system_id"""
    try:
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)
        maquinas = await client.machines.list()

        for m in maquinas:
            m_full = await client.machines.get(m.system_id)
            if (m_full.hostname.lower() == identificador.lower() or
                m_full.system_id.lower() == identificador.lower()):
                return m_full
        return None
    except Exception as e:
        print(f"Error buscando máquina: {e}")
        return None

# ==========================
# Funciones para Dashboard
# ==========================

async def obtener_metricas_dashboard():
    """Obtiene métricas completas para el dashboard"""
    try:
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)
        machines = await client.machines.list()

        metricas = {
            "resumen": await obtener_resumen_general(client, machines),
            "maquinas": await obtener_detalle_maquinas(client, machines),
            "red": await obtener_metricas_red(client),
            "alertas": await obtener_alertas_activas(client, machines),
            "rendimiento": await obtener_metricas_rendimiento(client, machines)
        }

        return serializar_objeto_simple(metricas)

    except Exception as e:
        print(f"Error obteniendo métricas del dashboard: {e}")
        return {
            "resumen": {},
            "maquinas": [],
            "red": {},
            "alertas": [],
            "rendimiento": {},
            "error": str(e)
        }

async def obtener_resumen_general(client, machines):
    """Obtiene resumen general del sistema"""
    try:
        total_maquinas = len(machines)
        maquinas_encendidas = 0
        maquinas_apagadas = 0
        total_ram = 0
        total_cpu = 0
        total_almacenamiento = 0

        for m in machines:
            m_full = await client.machines.get(m.system_id)
            power_state = m_full._data.get("power_state", "unknown")

            if power_state == "on":
                maquinas_encendidas += 1
            elif power_state == "off":
                maquinas_apagadas += 1

            # Recursos
            total_ram += m_full._data.get("memory", 0)
            total_cpu += m_full._data.get("cpu_count", 0)
            total_almacenamiento += m_full._data.get("storage", 0)

        return {
            "total_maquinas": total_maquinas,
            "maquinas_encendidas": maquinas_encendidas,
            "maquinas_apagadas": maquinas_apagadas,
```

```
            "maquinas_desconocidas": total_maquinas - maquinas_encendidas - maquinas_apagadas,
            "total_ram_gb": round(total_ram / 1024, 1),
            "total_cpu_cores": total_cpu,
            "total_almacenamiento_gb": round(total_almacenamiento / 1024, 1),
            "timestamp": datetime.now().isoformat()
        }
    except Exception as e:
        print(f"Error en resumen general: {e}")
        return {}

async def obtener_detalle_maquinas(client, machines):
    """Obtiene detalle de todas las máquinas"""
    try:
        detalle_maquinas = []
        
        for m in machines:
            try:
                m_full = await client.machines.get(m.system_id)
                
                power_state = m_full._data.get("power_state", "unknown")
                status_name = m_full.status_name
                
                # Calcular estado de salud
                salud = "healthy"
                if status_name in ["Failed", "Error"]:
                    salud = "critical"
                elif status_name in ["Deploying", "Commissioning"]:
                    salud = "warning"
                elif power_state == "unknown":
                    salud = "unknown"
                
                # Extraer información de forma segura
                zona_info = "default"
                pool_info = "default"
                
                try:
                    if m_full.zone:
                        zona_info = getattr(m_full.zone, "name", "default")
                except:
                    pass
                    
                try:
                    if m_full.pool:
                        pool_info = getattr(m_full.pool, "name", "default")
                except:
                    pass
                
                # Obtener IP de forma segura
                ip_principal = "Sin IP"
                try:
                    if m_full.ip_addresses and len(m_full.ip_addresses) > 0:
                        ip_principal = m_full.ip_addresses[0]
                except:
                    pass
                
                detalle_maquinas.append({
                    "hostname": m_full.hostname,
                    "system_id": m_full.system_id,
                    "power_state": power_state,
                    "status": status_name,
                    "salud": salud,
                    "ip": ip_principal,
                    "ram_gb": round(m_full._data.get("memory", 0) / 1024) if m_full._data.get("memory") else 0,
                    "almacenamiento_gb": round(m_full._data.get("storage", 0) / 1024, 1) if m_full._data.get("storage"
                    "cpu_cores": m_full._data.get("cpu_count", 0),
                    "so": f"{m_full.osystem} {m_full.distro_series}" if m_full.osystem else "No SO",
                    "zona": zona_info,
                    "pool": pool_info,
                    "ultima_actualizacion": datetime.now().isoformat()
                })
                
            except Exception as e:
                print(f"Error procesando máquina {m.system_id}: {e}")
                detalle_maquinas.append({
                    "hostname": f"Error-{m.system_id}",
                    "system_id": m.system_id,
                    "power_state": "unknown",
```

```
                    "status": "Error",
                    "salud": "critical",
                    "ip": "Error",
                    "ram_gb": 0,
                    "almacenamiento_gb": 0,
                    "cpu_cores": 0,
                    "so": "Error al cargar",
                    "zona": "default",
                    "pool": "default",
                    "ultima_actualizacion": datetime.now().isoformat(),
                    "error": str(e)
                })

        return detalle_maquinas
    except Exception as e:
        print(f"Error obteniendo detalle de máquinas: {e}")
        return []

async def obtener_metricas_red(client):
    """Obtiene métricas de red"""
    try:
        subnets = await client.subnets.list()

        metricas_red = {
            "total_subredes": len(subnets),
            "subredes": [],
            "ips_utilizadas": 0,
            "ips_disponibles": 0
        }

        for subnet in subnets:
            subnet_info = {
                "nombre": str(getattr(subnet, "name", "Sin nombre")),
                "cidr": str(getattr(subnet, "cidr", "Desconocido")),
                "vlan": "No asignada",
                "space": "default",
                "gateway": str(getattr(subnet, "gateway_ip", "No configurado"))
            }

            try:
                vlan_obj = getattr(subnet, "vlan", None)
                if vlan_obj:
                    vlan_id = str(getattr(vlan_obj, "id", "N/A"))
                    vlan_name = str(getattr(vlan_obj, "name", "Sin nombre"))
                    vlan_vid = str(getattr(vlan_obj, "vid", "N/A"))

                    subnet_info["vlan"] = f"{vlan_name} (VID: {vlan_vid}, ID: {vlan_id})"
            except Exception as e:
                print(f"Error procesando VLAN: {e}")
                subnet_info["vlan"] = "Error al obtener VLAN"

            try:
                space_obj = getattr(subnet, "space", None)
                if space_obj:
                    space_name = str(getattr(space_obj, "name", "default"))
                    subnet_info["space"] = space_name
            except Exception as e:
                print(f"Error procesando space: {e}")

            metricas_red["subredes"].append(subnet_info)

        return metricas_red
    except Exception as e:
        print(f"Error obteniendo métricas de red: {e}")
        return {
            "total_subredes": 0,
            "subredes": [],
            "ips_utilizadas": 0,
            "ips_disponibles": 0,
            "error": str(e)
        }

async def obtener_alertas_activas(client, machines):
    """Identifica alertas activas en el sistema"""
    try:
        alertas = []

```

```
        for m in machines:■
            m_full = await client.machines.get(m.system_id)■
            status_name = m_full.status_name■
            power_state = m_full._data.get("power_state", "unknown")■
            ■
            if status_name == "Failed":■
                alertas.append({■
                    "tipo": "critical",■
                    "maquina": m_full.hostname,■
                    "mensaje": "Máquina en estado Failed",■
                    "timestamp": datetime.now().isoformat()■
                })■
            elif status_name == "Error":■
                alertas.append({■
                    "tipo": "critical", ■
                    "maquina": m_full.hostname,■
                    "mensaje": "Máquina en estado Error",■
                    "timestamp": datetime.now().isoformat()■
                })■
            elif power_state == "unknown":■
                alertas.append({■
                    "tipo": "warning",■
                    "maquina": m_full.hostname,■
                    "mensaje": "Estado de energía desconocido",■
                    "timestamp": datetime.now().isoformat()■
                })■
        ■
        return alertas■
    except Exception as e:■
        print(f"Error obteniendo alertas: {e}")■
        return []■
■
async def obtener_metricas_rendimiento(client, machines):■
    """Obtiene métricas de rendimiento"""■
    try:■
        maquinas_encendidas = 0■
        for m in machines:■
            m_full = await client.machines.get(m.system_id)■
            if m_full._data.get("power_state") == "on":■
                maquinas_encendidas += 1■
        ■
        return {■
            "uso_cpu_promedio": 0,■
            "uso_memoria_promedio": 0,■
            "io_disponible": "Normal",■
            "latencia_red": "Baja",■
            "maquinas_activas": maquinas_encendidas■
        }■
    except Exception as e:■
        print(f"Error obteniendo métricas de rendimiento: {e}")■
        return {}■
■
# ==========================■
# Funciones de utilidad para búsqueda■
# ==========================■
■
def extraer_identificador_maquina(pregunta, maquinas_texto):■
    """Extrae el identificador de máquina de una pregunta"""■
    pregunta_lower = pregunta.lower()■
    identificador = None■
    ■
    # Buscar por nombre de máquina■
    for linea in maquinas_texto.split('\n'):■
        if 'MÁQUINA:' in linea:■
            partes = linea.split('(')■
            if len(partes) > 1:■
                nombre_maquina = partes[0].replace('■ MÁQUINA:', '').strip()■
                system_id = partes[1].replace(')', '').strip()■
                ■
                if nombre_maquina.lower() in pregunta_lower:■
                    identificador = nombre_maquina■
                    break■
                elif system_id.lower() in pregunta_lower:■
                    identificador = system_id■
                    break■
    ■
    # Buscar por IP■
```

```
if not identificador:■
    ip_pattern = r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b'■
    ips = re.findall(ip_pattern, pregunta)■
    if ips:■
        return ips[0]■
■
return identificador
```

Archivo 5: ./services/gemini_service.py

```
import google.generativeai as genai■
from config import GEMINI_API_KEY■
■
genai.configure(api_key=GEMINI_API_KEY)■
model = genai.GenerativeModel('gemini-2.0-flash')■
■
def generar_respuesta_gemini(prompt: str) -> str:■
    try:■
        respuesta = model.generate_content(prompt)■
        return respuesta.text.strip()■
    except Exception as e:■
        return f"■ Error al generar respuesta: {e}"
```

Archivo 6: ./services/telegram_service.py

```
import requests■
from config import TELEGRAM_BOT_TOKEN, TELEGRAM_CHAT_ID■
■
def enviar_notificacion_telegram(mensaje):■
    try:■
        if not TELEGRAM_BOT_TOKEN or not TELEGRAM_CHAT_ID:■
            print("■■ Configuración de Telegram no completada")■
            return False■
            ■
        url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"■
        payload = {■
            "chat_id": TELEGRAM_CHAT_ID,■
            "text": mensaje,■
            "parse_mode": "HTML"■
        }■
        response = requests.post(url, json=payload, timeout=10)■
        return response.status_code == 200■
    except Exception as e:■
        print(f"Error enviando notificación a Telegram: {e}")■
        return False
```