■ Proyecto Flask - Código Python (Parte 1)

Total de archivos: 8

Archivo 1: ./app.py

```
from flask import Flask, jsonify■
from flask_cors import CORS■
import threading■
from datetime import datetime■
from config import TELEGRAM_BOT_TOKEN, TELEGRAM_CHAT_ID■
from models.monitor import MonitorMaquinas■
from services.telegram_service import enviar_notificacion_telegram■
# Importar rutas■
from routes.chat_routes import setup_chat_routes■
from routes.monitor_routes import setup_monitor_routes■
from routes.dashboard_routes import setup_dashboard_routes■
from routes.commissioning_routes import setup_commissioning_routes■
from routes.deploy_routes import setup_deploy_routes■
app = Flask(__name__)■
CORS(app)■
■
# Instancia global del monitor■
monitor = MonitorMaquinas()■
■
# Configurar rutas■
setup_chat_routes(app)■
setup_monitor_routes(app, monitor)■
setup_dashboard_routes(app)■
setup_commissioning_routes(app)■
setup_deploy_routes(app)■
■
@app.route('/')■
def index(): ■
    return jsonify({■
        "mensaje": "MAAS Bot API está funcionando correctamente",■
        "version": "2.0",■
        "endpoints": {■
            "/preguntar": "POST - Enviar preguntas al asistente",■
            "/monitor/start": "POST - Iniciar monitoreo", ■
            "/monitor/stop": "POST - Detener monitoreo",■
            "/monitor/status": "GET - Estado del monitoreo",■
            "/dashboard/metricas": "GET - Métricas del dashboard",■
            "/health": "GET - Health check"■
        }■
    })■
■
@app.route('/health', methods=["GET"])■
def health_check(): ■
    return jsonify({■
        "status": "healthy",■
        "timestamp": datetime.now().isoformat(),■
        "service": "MAAS Bot API"■
    })■
■
def iniciar_aplicacion(): ■
    print("Iniciando aplicación MAAS Bot...")■
    ■
    # Notificación de inicio■
    mensaje_inicio = f"<b>MAAS Bot Iniciado</b>\nSistema de monitorización listo\n{datetime.now().strftime('%Y-%m-%d %
    threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_inicio)).start()■
■
if __name__ == "__main__":■
    iniciar_aplicacion()■
    print("Servidor Flask iniciado en http://0.0.0.0:5000")■
    app.run(host="0.0.0.0", port=5000, debug=False)
```

Archivo 2: ./config.py

```
# =========================■
# Configuración■
# =========================■
MAAS_URL = "http://172.16.25.2:5240/MAAS"■
MAAS_API_KEY = "E9nebNWw3WhSejrkAL:Av2xxeCgHq2jeGL2rG:skcKZQp85vMdya2WubtERYXhMxf7pTty"■
GEMINI_API_KEY = "AIzaSyCVmwiNmBqMaYQrvGDMBzPY_GJwrDNynt4"■
TELEGRAM_BOT_TOKEN = "8436841267:AAF5oYG_FiKvDNi-vKGh_JjL4X_v3ReQUHo"■
TELEGRAM_CHAT_ID = "5786912071"■
■
CORTESIAS = ["gracias", "muchas gracias", "ok gracias", "thank you", "ok", "perfecto"]
```

Archivo 3: ./export_code_to_pdf.py

```
import os■
from reportlab.lib.pagesizes import A4■
from reportlab.pdfgen import canvas■
■
# ■ Ruta del proyecto Flask (misma carpeta que app.py)■
PROJECT_PATH = "./"■
■
# ■ Solo archivos Python■
VALID_EXTENSIONS = (".py",)■
■
# ■ Carpetas a ignorar■
IGNORE_DIRS = {"maas-env"}■
■
■
def obtener_archivos(directorio):■
    """Recorre el directorio de forma recursiva y obtiene todos los archivos .py, ignorando carpetas especificadas."""
    archivos_validos = []■
    for raiz, subdirs, archivos in os.walk(directorio):■
        # Filtrar subdirectorios que queremos ignorar■
        subdirs[:] = [d for d in subdirs if d not in IGNORE_DIRS]■
■
        for archivo in archivos:■
            if archivo.endswith(VALID_EXTENSIONS):■
                archivos_validos.append(os.path.join(raiz, archivo))■
    return archivos_validos■
■
■
def escribir_pdf(nombre_pdf, titulo, archivos):■
    """Genera un PDF con el contenido de los archivos Python."""■
    pdf = canvas.Canvas(nombre_pdf, pagesize=A4)■
    ancho, alto = A4■
■
    pdf.setFont("Helvetica-Bold", 14)■
    pdf.drawCentredString(ancho / 2, alto - 30, f"■ {titulo}")■
    pdf.setFont("Helvetica", 10)■
    pdf.drawString(30, alto - 50, f"Total de archivos: {len(archivos)}")■
■
    for i, ruta in enumerate(archivos, 1):■
        pdf.showPage()■
        pdf.setFont("Helvetica-Bold", 12)■
        pdf.drawString(30, alto - 40, f"Archivo {i}: {ruta}")■
        pdf.setFont("Courier", 8)■
■
        try:■
            with open(ruta, "r", encoding="utf-8", errors="ignore") as f:■
                contenido = f.readlines()■
        except Exception as e:■
            contenido = [f"■■ No se pudo leer el archivo: {e}"]■
■
        y = alto - 60■
        for linea in contenido:■
            if y < 40:  # Salto de página si se acaba el espacio■
                pdf.showPage()■
                y = alto - 40■
                pdf.setFont("Courier", 8)■
            pdf.drawString(30, y, linea[:130])  # Cortar líneas largas■
            y -= 10■
■
    pdf.save()■
    print(f"■ PDF generado: {nombre_pdf}")■
■
■
def main():■
    archivos = obtener_archivos(PROJECT_PATH)■
    mitad = (len(archivos) + 1) // 2■
    primera_mitad = archivos[:mitad]■
    segunda_mitad = archivos[mitad:]■
■
    escribir_pdf("Proyecto_Flask_Backend_Parte1.pdf",■
                 "Proyecto Flask - Código Python (Parte 1)", primera_mitad)■
    escribir_pdf("Proyecto_Flask_Backend_Parte2.pdf",■
                 "Proyecto Flask - Código Python (Parte 2)", segunda_mitad)■
■
■
if __name__ == "__main__":■
```

```
main()■
```

Archivo 4: ./explorar_api_maas.py

```
#!/usr/bin/env python3■
■
import asyncio■
from maas.client import connect■
from config import MAAS_URL, MAAS_API_KEY■
■
async def listar_maquinas_ready():■
    """Lista solo las máquinas en estado Ready para deploy"""■
    client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
    machines = await client.machines.list()■
    ■
    print("\n" + "="*80)■
    print("■ MÁQUINAS LISTAS PARA DEPLOY (Estado: Ready)")■
    print("="*80)■
    ■
    maquinas_ready = []■
    ■
    for i, maquina in enumerate(machines, 1):■
        estado = getattr(maquina, 'status_name', f'Código: {maquina.status}')■
        ■
        # Solo incluir máquinas en estado Ready■
        if estado == 'Ready':■
            # Obtener información adicional útil para deploy■
            sistema_operativo = getattr(maquina, 'osystem', 'No definido')■
            arquitectura = getattr(maquina, 'architecture', 'No definida')■
            ■
            print(f"{i}. {maquina.hostname} | SO: {sistema_operativo} | Arquitectura: {arquitectura} | ID: {maquina.sy
            maquinas_ready.append(maquina)■
    ■
    return maquinas_ready■
■
async def hacer_deploy(maquina):■
    """Ejecuta deploy en la máquina seleccionada"""■
    print(f"\n■ Preparando deploy en: {maquina.hostname}")■
    print(f"■ Estado actual: {getattr(maquina, 'status_name', 'Desconocido')}")■
    ■
    try:■
        # Verificar si la máquina está en estado Ready■
        estado_actual = getattr(maquina, 'status_name', 'Desconocido')■
        ■
        if estado_actual != 'Ready':■
            print(f"■ La máquina no está en estado Ready para deploy")■
            print(f"■ Estado actual: {estado_actual}")■
            print(f"■ Estado requerido: Ready")■
            return■
        ■
        # Mostrar información de la máquina■
        sistema_operativo = getattr(maquina, 'osystem', 'No definido')■
        arquitectura = getattr(maquina, 'architecture', 'No definida')■
        memoria = getattr(maquina, 'memory', 0) / 1024  # Convertir a GB■
        ■
        print(f"\n■ Información de la máquina:")■
        print(f"   • Hostname: {maquina.hostname}")■
        print(f"   • Sistema Operativo: {sistema_operativo}")■
        print(f"   • Arquitectura: {arquitectura}")■
        print(f"   • Memoria: {memoria:.1f} GB")■
        print(f"   • CPU: {getattr(maquina, 'cpu_count', 'N/A')} cores")■
        ■
        # Confirmar deploy■
        print(f"\n■■  ¿Estás seguro de hacer DEPLOY en {maquina.hostname}?")■
        confirmar = input("   (s/n): ").lower().strip()■
        ■
        if confirmar != 's':■
            print("■ Deploy cancelado")■
            return■
        ■
        # Configurar opciones de deploy■
        print("\n■ Opciones de deploy:")■
        print("1. Deploy estándar")■
        print("2. Deploy con usuario SSH")■
        print("3. Deploy personalizado")■
        ■
        opcion = input("   Elige opción (1-3, Enter=1): ").strip() or "1"■
        ■
        if opcion == "1":■
```

```
            # Deploy estándar■
            resultado = await maquina.deploy(wait=False)■
        elif opcion == "2":■
            # Deploy con usuario SSH■
            usuario_ssh = input("   Usuario SSH: ").strip()■
            if not usuario_ssh:■
                usuario_ssh = "ubuntu"  # Default■
            ■
            resultado = await maquina.deploy(■
                user_data=None,■
                distro_series=None,■
                hwe_kernel=None,■
                wait=False,■
                install_rackd=False■
            )■
        elif opcion == "3":■
            # Deploy personalizado■
            usuario_ssh = input("   Usuario SSH (Enter para default 'ubuntu'): ").strip() or "ubuntu"■
            serie_distro = input("   Serie distro (Ej: focal, jammy - Enter para default): ").strip() or None■
            kernel = input("   Kernel HWE (Ej: hwe-22.04 - Enter para default): ").strip() or None■
            ■
            resultado = await maquina.deploy(■
                user_data=None,■
                distro_series=serie_distro,■
                hwe_kernel=kernel,■
                wait=False,■
                install_rackd=False■
            )■
        else:■
            print("■ Opción no válida")■
            return■
        ■
        print("■ Deploy iniciado correctamente")■
        ■
        # Monitorear progreso■
        print("\n■ Monitoreando progreso del deploy...")■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        ■
        for i in range(60):  # 10 minutos máximo (deploy suele tomar más tiempo)■
            await asyncio.sleep(10)■
            maquina_actualizada = await client.machines.get(maquina.system_id)■
            estado_actual = getattr(maquina_actualizada, 'status_name', f'Código: {maquina_actualizada.status}')■
            print(f"   [{i+1}/60] Estado: {estado_actual}")■
            ■
            # Estados finales■
            if estado_actual in ['Deployed', 'Failed', 'Broken']:■
                if estado_actual == 'Deployed':■
                    print("■ DEPLOY COMPLETADO EXITOSAMENTE!")■
                    # Obtener IP si está disponible■
                    direcciones_ip = getattr(maquina_actualizada, 'ip_addresses', [])■
                    if direcciones_ip:■
                        print(f"■ Dirección IP: {', '.join(direcciones_ip)}")■
                else:■
                    print(f"■■ Deploy terminó con estado: {estado_actual}")■
                break■
            ■
            # Si sigue en deploy■
            if estado_actual in ['Deploying', 'Allocating']:■
                continue■
                ■
        else:■
            print("■ Deploy aún en progreso después de 10 minutos")■
            ■
    except Exception as e:■
        print(f"■ Error durante deploy: {e}")■
■
async def main():■
    """Función principal - Menú interactivo para deploy"""■
    try:■
        print("■ DEPLOY DE MÁQUINAS MAAS")■
        print("="*80)■
        ■
        while True:■
            # Listar solo máquinas en estado Ready■
            maquinas = await listar_maquinas_ready()■
            ■
            if not maquinas:■
```

```
                print("■ No hay máquinas disponibles para deploy")■
                print("■ Las máquinas deben estar en estado 'Ready'")■
                return■
            ■
            # Seleccionar máquina■
            print("\n" + "-"*80)■
            seleccion = input("■ Elige una máquina (número) o 'q' para salir: ").strip()■
            ■
            if seleccion.lower() == 'q':■
                print("■ ¡Hasta luego!")■
                break■
            ■
            try:■
                indice = int(seleccion) - 1■
                if 0 <= indice < len(maquinas):■
                    maquina_seleccionada = maquinas[indice]■
                    await hacer_deploy(maquina_seleccionada)■
                else:■
                    print("■ Número de máquina no válido")■
            except ValueError:■
                print("■ Entrada no válida. Ingresa un número.")■
            ■
            # Preguntar si quiere continuar■
            print("\n" + "-"*80)■
            continuar = input("¿Quieres elegir otra máquina? (s/n): ").lower().strip()■
            if continuar != 's':■
                print("■ ¡Hasta luego!")■
                break■
            ■
    except Exception as e:■
        print(f"■ Error general: {e}")■
        import traceback■
        traceback.print_exc()■
■
if __name__ == "__main__":■
    asyncio.run(main())
```

Archivo 5: ./telegram_bot_standalone.py

```
import asyncio■
import logging■
import signal■
import sys■
from telegram import Update■
from telegram.ext import Application, CommandHandler, MessageHandler, filters, CallbackContext■
from config import TELEGRAM_BOT_TOKEN■
from services.chat_service import responder_pregunta■
from models.monitor import MonitorMaquinas■
■
# Configurar logging■
logging.basicConfig(■
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',■
    level=logging.INFO■
)■
logger = logging.getLogger(__name__)■
■
class TelegramBotStandalone:■
    def __init__(self):■
        self.monitor = MonitorMaquinas()■
        self.application = None■
■
    async def start(self, update: Update, context: CallbackContext) -> None:■
        """Envía un mensaje cuando se emite el comando /start"""■
        user = update.effective_user■
        welcome_message = (■
            f"■ ¡Hola {user.first_name}! Soy tu asistente MAAS Bot\n\n"■
            "Puedo ayudarte a:\n"■
            "• Consultar el estado de las máquinas\n"■
            "• Encender y apagar máquinas\n"■
            "• Monitorear cambios en el sistema\n"■
            "• Consultar información de red\n\n"■
            "Ejemplos de comandos:\n"■
            "• '¿Qué máquinas están encendidas?'\n"■
            "• 'Enciende la máquina X'\n"■
            "• 'Apaga la máquina Y'\n"■
            "• 'Muestra información de las subredes'\n\n"■
            "Usa /monitor para controlar el monitoreo automático"■
        )■
        await update.message.reply_text(welcome_message)■
■
    async def monitor_command(self, update: Update, context: CallbackContext) -> None:■
        """Controla el monitoreo automático"""■
        if not self.monitor.monitoreo_activo:■
            # Iniciar monitoreo■
            asyncio.create_task(self.monitor.iniciar_monitoreo())■
            await update.message.reply_text(■
                "■ Monitoreo iniciado. Recibirás notificaciones de cambios en las máquinas."■
            )■
        else:■
            # Detener monitoreo■
            self.monitor.detener_monitoreo()■
            await update.message.reply_text("■ Monitoreo detenido.")■
■
    async def status_command(self, update: Update, context: CallbackContext) -> None:■
        """Muestra el estado actual del monitoreo"""■
        status = "■ ACTIVO" if self.monitor.monitoreo_activo else "■ INACTIVO"■
        maquinas_monitoreadas = len(self.monitor.estados_anteriores)■
        ■
        status_message = (■
            f"■ Estado del Sistema:\n"■
            f"Monitoreo: {status}\n"■
            f"Máquinas monitoreadas: {maquinas_monitoreadas}\n"■
            f"Intervalo: {self.monitor.intervalo} segundos"■
        )■
        await update.message.reply_text(status_message)■
■
    async def handle_message(self, update: Update, context: CallbackContext) -> None:■
        """Procesa mensajes de texto usando la misma lógica del chat web"""■
        user_message = update.message.text■
        ■
        if not user_message.strip():■
            await update.message.reply_text("Por favor, envía un mensaje válido.")■
            return■
■
```

```
        try:■
            # Mostrar indicador de escritura■
            await context.bot.send_chat_action(■
                chat_id=update.effective_chat.id, ■
                action="typing"■
            )■
            ■
            # Usar la misma función del chat web■
            respuesta = await responder_pregunta(user_message)■
            ■
            # Enviar respuesta■
            await update.message.reply_text(respuesta, parse_mode='HTML')■
            ■
        except Exception as e:■
            error_message = f"■ Error procesando tu mensaje: {str(e)}"■
            await update.message.reply_text(error_message)■
            logger.error(f"Error en Telegram bot: {e}")■
■
    async def help_command(self, update: Update, context: CallbackContext) -> None:■
        """Muestra la ayuda"""■
        help_text = (■
            "■ Comandos disponibles:\n\n"■
            "/start - Iniciar el bot\n"■
            "/help - Mostrar esta ayuda\n"■
            "/monitor - Iniciar/detener monitoreo automático\n"■
            "/status - Estado del sistema\n\n"■
            "■ También puedes enviar mensajes como:\n"■
            "• 'lista las máquinas'\n"■
            "• 'enciende servidor01'\n"■
            "• 'apaga 172.16.25.201'\n"■
            "• 'muestra las subredes'\n"■
            "• '¿cuánta RAM tiene la máquina X?'"■
        )■
        await update.message.reply_text(help_text)■
■
    async def run_bot(self):■
        """Inicia el bot de Telegram"""■
        try:■
            print("■ Iniciando bot de Telegram...")■
            ■
            # Crear la aplicación■
            self.application = (■
                Application.builder()■
                .token(TELEGRAM_BOT_TOKEN)■
                .build()■
            )■
            ■
            # Añadir handlers■
            self.application.add_handler(CommandHandler("start", self.start))■
            self.application.add_handler(CommandHandler("help", self.help_command))■
            self.application.add_handler(CommandHandler("monitor", self.monitor_command))■
            self.application.add_handler(CommandHandler("status", self.status_command))■
            self.application.add_handler(■
                MessageHandler(filters.TEXT & ~filters.COMMAND, self.handle_message)■
            )■
            ■
            print("■ Bot de Telegram configurado correctamente")■
            ■
            # Iniciar el bot■
            await self.application.initialize()■
            await self.application.start()■
            print("■ Bot de Telegram iniciado, comenzando polling...")■
            ■
            # Ejecutar polling■
            await self.application.updater.start_polling()■
            ■
            # Mantener el bot corriendo■
            print("■ Bot de Telegram está ahora activo y escuchando mensajes...")■
            ■
            # Esperar indefinidamente■
            await asyncio.Event().wait()■
            ■
        except Exception as e:■
            print(f"■ Error en el bot de Telegram: {e}")■
            raise■
■
    async def stop_bot(self):■
```

```
        """Detiene el bot de Telegram correctamente"""■
        if self.application:■
            print("■ Deteniendo bot de Telegram...")■
            await self.application.updater.stop()■
            await self.application.stop()■
            await self.application.shutdown()■
            print("■ Bot de Telegram detenido correctamente")■
■
def signal_handler(signum, frame):■
    """Maneja señales de terminación"""■
    print(f"\n■ Señal {signum} recibida, deteniendo bot...")■
    sys.exit(0)■
■
async def main():■
    bot = TelegramBotStandalone()■
    ■
    # Registrar manejador de señales■
    signal.signal(signal.SIGINT, signal_handler)■
    signal.signal(signal.SIGTERM, signal_handler)■
    ■
    try:■
        await bot.run_bot()■
    except KeyboardInterrupt:■
        print("\n■ Interrupción por teclado recibida")■
    except Exception as e:■
        print(f"■ Error fatal: {e}")■
    finally:■
        await bot.stop_bot()■
■
if __name__ == "__main__":■
    # Ejecutar el bot■
    asyncio.run(main())
```

Archivo 6: ./routes/dashboard_routes.py

```
from flask import jsonify■
import asyncio■
from services.maas_client import obtener_metricas_dashboard■
■
def setup_dashboard_routes(app):■
    """Configura las rutas del dashboard"""■
    ■
    @app.route("/dashboard/metricas", methods=["GET"])■
    def obtener_metricas_dashboard_endpoint():■
        """Endpoint para obtener todas las métricas del dashboard"""■
        try:■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            metricas = loop.run_until_complete(obtener_metricas_dashboard())■
            return jsonify(metricas)■
        except Exception as e:■
            print(f"Error en endpoint /dashboard/metricas: {e}")■
            return jsonify({■
                "resumen": {},■
                "maquinas": [],■
                "red": {},■
                "alertas": [],■
                "rendimiento": {},■
                "error": str(e)■
            }), 500■
■
    @app.route("/dashboard/maquinas", methods=["GET"])■
    def obtener_maquinas_dashboard_endpoint():■
        """Endpoint para obtener detalle de máquinas"""■
        try:■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            metricas = loop.run_until_complete(obtener_metricas_dashboard())■
            return jsonify(metricas.get("maquinas", []))■
        except Exception as e:■
            print(f"Error en endpoint /dashboard/maquinas: {e}")■
            return jsonify({"error": str(e), "maquinas": []}), 500■
■
    @app.route("/dashboard/alertas", methods=["GET"])■
    def obtener_alertas_endpoint():■
        """Endpoint para obtener alertas activas"""■
        try:■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            metricas = loop.run_until_complete(obtener_metricas_dashboard())■
            return jsonify(metricas.get("alertas", []))■
        except Exception as e:■
            print(f"Error en endpoint /dashboard/alertas: {e}")■
            return jsonify({"error": str(e), "alertas": []}), 500
```

Archivo 7: ./routes/deploy_routes.py

```
from flask import jsonify, request■
import asyncio■
from services.maas_client import listar_maquinas_para_deploy, ejecutar_deploy■
■
def setup_deploy_routes(app):■
    """Configura las rutas para deploy"""■
■
    @app.route("/deploy/maquinas", methods=["GET"])■
    def listar_maquinas_deploy():■
        """Endpoint para listar máquinas disponibles para deploy"""■
        try:■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            maquinas = loop.run_until_complete(listar_maquinas_para_deploy())■
            return jsonify({■
                'maquinas': maquinas,■
                'total': len(maquinas)■
            })■
        except Exception as e:■
            return jsonify({'error': str(e)}), 500■
■
    @app.route("/deploy/ejecutar", methods=["POST"])■
    def ejecutar_deploy_endpoint():■
        """Endpoint para ejecutar deploy en una máquina"""■
        try:■
            data = request.get_json()■
            if not data or 'system_id' not in data:■
                return jsonify({'error': 'Se requiere system_id'}), 400■
■
            system_id = data['system_id']■
            opciones = data.get('opciones', {})■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            resultado = loop.run_until_complete(ejecutar_deploy(system_id, opciones))■
■
            return jsonify(resultado)■
■
        except Exception as e:■
            return jsonify({'error': str(e)}), 500
```

Archivo 8: ./routes/commissioning_routes.py

```
from flask import jsonify, request■
import asyncio■
from services.maas_client import listar_maquinas_para_commissioning, ejecutar_commissioning, obtener_estado_commission
■
def setup_commissioning_routes(app):■
    """Configura las rutas para commissioning"""■
    ■
    @app.route("/commissioning/maquinas", methods=["GET"])■
    def listar_maquinas_commissioning():■
        """Endpoint para listar máquinas disponibles para commissioning"""■
        try:■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            maquinas = loop.run_until_complete(listar_maquinas_para_commissioning())■
            return jsonify({■
                'maquinas': maquinas,■
                'total': len(maquinas)■
            })■
        except Exception as e:■
            return jsonify({'error': str(e)}), 500■
    ■
    @app.route("/commissioning/ejecutar", methods=["POST"])■
    def ejecutar_commissioning_endpoint():■
        """Endpoint para ejecutar commissioning en una máquina"""■
        try:■
            data = request.get_json()■
            if not data or 'system_id' not in data:■
                return jsonify({'error': 'Se requiere system_id'}), 400■
            ■
            system_id = data['system_id']■
            opciones = data.get('opciones', {})■
            ■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            resultado = loop.run_until_complete(ejecutar_commissioning(system_id, opciones))■
            ■
            return jsonify(resultado)■
            ■
        except Exception as e:■
            return jsonify({'error': str(e)}), 500■
    ■
    @app.route("/commissioning/estado/<system_id>", methods=["GET"])■
    def estado_commissioning(system_id):■
        """Endpoint para obtener estado del commissioning"""■
        try:■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            estado = loop.run_until_complete(obtener_estado_commissioning(system_id))■
            return jsonify(estado)■
        except Exception as e:■
            return jsonify({'error': str(e)}), 500
```