Source: C:\Users\Admin\Desktop\CMC\ms1_emailIngestion\api\ms1_apiHanlder.py

```
"""
api_service.py
HTTP API ■i■u khi■n Email Ingestion Microservice
Ch■y trên port riêng (8000) - không conflict v■i webhook (8100)
"""
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import Optional
from enum import Enum
from main_orchestrator import orchestrator
from core.session_manager import TriggerMode

app = FastAPI(
    title="Email Ingestion Control API",
    description="API ■i■u khi■n email ingestion microservice",
    version="1.0.0"
)

class PollingModeEnum(str, Enum):
    manual = "manual"
    scheduled = "scheduled"

class StartSessionRequest(BaseModel):
    """Request ■■ start session"""
    polling_mode: PollingModeEnum = PollingModeEnum.scheduled
    polling_interval: int = 300
    enable_webhook: bool = True

class StopSessionRequest(BaseModel):
    """Request ■■ stop session"""
    reason: Optional[str] = "user_requested"

@app.get("/")
async def root():
    """Root endpoint"""
    return {
        "service": "Email Ingestion Microservice",
        "version": "1.0.0",
        "status": "running"
    }

@app.get("/health")
async def health_check():
    """Health check"""
    return {"status": "healthy"}

@app.post("/session/start")
async def start_session(request: StartSessionRequest):
    """
    Kh■i ■■ng phiên làm vi■c m■i
    """
    try:
        mode = TriggerMode.MANUAL if request.polling_mode == "manual" else TriggerMode.SCHEDULED

        success = orchestrator.start_session(
            polling_mode=mode,
            polling_interval=request.polling_interval,
            enable_webhook=request.enable_webhook
        )

        if not success:
            raise HTTPException(
                status_code=400,
                detail="Failed to start session. Check if session is already running."
            )

        status = orchestrator.get_status()
        return {
            "success": True,
            "message": "Session started successfully",
            "session_id": status["session"]["session_id"],
            "status": status
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/session/stop")
async def stop_session(request: StopSessionRequest):
    """
    D■ng phiên làm vi■c hi■n t■i
    """
    try:
        if not orchestrator.running:
            raise HTTPException(status_code=400, detail="No active session")

        orchestrator.stop_session(reason=request.reason)
        return {
            "success": True,
            "message": "Session stopped successfully"
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.get("/session/status")
async def get_session_status():
    """
    L■y tr■ng thái phiên làm vi■c hi■n t■i
    """
    try:
        status = orchestrator.get_status()
        return status
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/polling/trigger")
async def trigger_manual_poll():
    """
    Trigger m■t l■n polling th■ công
    """
    try:
        if not orchestrator.running:
            raise HTTPException(
                status_code=400,
                detail="No active session. Start a session first."
            )

        result = orchestrator.trigger_manual_poll()

        if result.get("status") == "error":
            raise HTTPException(status_code=500, detail=result.get("error"))

        return result
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@app.get("/metrics")
async def get_metrics():
    """
    L■y metrics t■ng quan
    """
    try:
        status = orchestrator.get_status()
        return {
            "session_active": orchestrator.running,
            "total_processed": status["session"]["processed_count"],
            "total_pending": status["session"]["pending_count"],
            "polling_errors": status["session"]["polling_errors"],
            "webhook_errors": status["session"]["webhook_errors"],
            "services": {
                "polling_active": status["polling"]["active"],
                "webhook_active": status["webhook"]["active"]
            },
            "timestamp": status["timestamp"]
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
if __name__ == "__main__":
```

```
import uvicorn
uvicorn.run(app, host="0.0.0.0", port=8000)
```