Source: C:\Users\Admin\Desktop\CMC\ms1_emailIngestion\core\webhook_service.py

```
"""
Webhook Service
X■ lý email theo c■ ch■ webhook v■i ngrok tunnel riêng bi■t
"""
import asyncio
import json
import requests
import threading
from datetime import datetime, timezone, timedelta
from typing import Optional, Dict
from pyngrok import ngrok
import psutil
import time
from core.session_manager import session_manager, SessionState
from core.queue_manager import get_email_queue
from core.token_manager import get_token

class WebhookService:
    """D■ch v■ webhook cho email notifications"""

    GRAPH_URL = "https://graph.microsoft.com/v1.0"
    WEBHOOK_PORT = 8100  # Port riêng cho webhook

    def __init__(self):
        self.active = False
        self.public_url: Optional[str] = None
        self.subscription_id: Optional[str] = None
        self.queue = get_email_queue()
        self.ngrok_tunnel = None
        self.error_count = 0
        self.max_errors = 5
        self.app = None
        self.server_process = None

    def start(self) -> bool:
        """Kh■i ■■ng webhook service"""
        if self.active:
            print(f"[WebhookService] Already active")
            return False

        try:
            print(f"[WebhookService] Starting on port {self.WEBHOOK_PORT}...")

            # Step 1: Kill existing processes
            self._kill_port_process(self.WEBHOOK_PORT)
            self._kill_existing_ngrok()

            # Step 2: Start ngrok tunnel
            self.public_url = self._start_ngrok()
            print(f"[WebhookService] Public URL: {self.public_url}")

            # Step 3: Start FastAPI server
            self._start_fastapi_server()
            time.sleep(3)  # ■■i server kh■i ■■ng

            # Step 4: Create subscription
            self.subscription_id = self._create_subscription()
            if not self.subscription_id:
                raise Exception("Failed to create subscription")

            print(f"[WebhookService] Subscription created: {self.subscription_id}")

            # Step 5: Start renewal watcher
            self._start_renewal_watcher()

            self.active = True
            self.error_count = 0

            print(f"[WebhookService] Started successfully")
            return True

        except Exception as e:
            print(f"[WebhookService] Start error: {e}")
            self.stop()
            return False

    def stop(self):
        """D■ng webhook service"""
        if not self.active:
            return

        print(f"[WebhookService] Stopping...")

        # Delete subscription
        if self.subscription_id:
            self._delete_subscription()

        # Stop FastAPI server
        if self.server_process:
            self.server_process.terminate()
            self.server_process = None

        # Close ngrok tunnel
        if self.ngrok_tunnel:
            ngrok.disconnect(self.ngrok_tunnel.public_url)
            self.ngrok_tunnel = None

        self.active = False
        print(f"[WebhookService] Stopped")

    def handle_notification(self, notification_data: Dict) -> Dict:
        """X■ lý notification t■ Microsoft Graph"""
        try:
            enqueued_count = 0
            notifications = notification_data.get("value", [])

            for notif in notifications:
                msg_id = notif.get("resourceData", {}).get("id")
                if not msg_id:
                    continue

                # Ki■m tra duplicate
                if self.queue.is_in_queue(msg_id) or session_manager.is_email_processed(msg_id):
                    print(f"[WebhookService] Skipping duplicate email: {msg_id}")
                    continue

                # Fetch email detail
                message = self._fetch_email_detail(msg_id)
                if message:
                    # Enqueue email for batch processing
                    enqueued_id = self.queue.enqueue(msg_id, message)
                    if enqueued_id:
                        session_manager.register_pending_email(msg_id)
                        enqueued_count += 1
                        print(f"[WebhookService] Enqueued email: {msg_id}")
                        self._mark_as_read(enqueued_id)  # Mark as read immediately

            # Reset error count khi thành công
            self.error_count = 0

            return {
                "status": "success",
                "enqueued": enqueued_count
            }

        except Exception as e:
            print(f"[WebhookService] Notification handling error: {e}")
            self.error_count += 1

            # Kích ho■t fallback n■u quá nhi■u l■i
            if self.error_count >= self.max_errors:
                self._activate_fallback()

            return {
                "status": "error",
                "error": str(e)
            }

    def _activate_fallback(self):
        """Kích ho■t fallback polling khi webhook l■i"""
        print(f"[WebhookService] Too many errors ({self.error_count}), activating fallback")

        session_manager.activate_fallback_polling(
            reason=f"webhook_errors_{self.error_count}"
        )

        # Import polling service ■■ kích ho■t
        from core.polling_service import polling_service, TriggerMode
        polling_service.start(mode=TriggerMode.FALLBACK, interval=300)

    def _fetch_email_detail(self, message_id: str) -> Optional[Dict]:
        """L■y chi ti■t email t■ Graph API"""
        token = get_token()
        headers = {"Authorization": f"Bearer {token}"}
        url = f"{self.GRAPH_URL}/me/messages/{message_id}"

        try:
            resp = requests.get(url, headers=headers, timeout=10)
            if resp.status_code == 200:
                return resp.json()
            return None
        except Exception as e:
            print(f"[WebhookService] Fetch email error: {e}")
            return None

    def _mark_as_read(self, message_id: str):
        """Mark a single email as read in a background thread."""
        token = get_token()
        headers = {
            "Authorization": f"Bearer {token}",
            "Content-Type": "application/json"
        }
        url = f"{self.GRAPH_URL}/me/messages/{message_id}"
        body = {"isRead": True}

        def do_patch():
            try:
                requests.patch(url, headers=headers, json=body, timeout=10)
                print(f"[WebhookService] ✓ Marked {message_id} as read.")
            except requests.exceptions.RequestException as e:
                print(f"[WebhookService] ERROR: Failed to mark {message_id} as read: {e}")

        # Run in a separate thread to not block the webhook response
        threading.Thread(target=do_patch, daemon=True).start()

    def _start_ngrok(self) -> str:
        """Kh■i ■■ng ngrok tunnel riêng cho webhook"""
        try:
            self.ngrok_tunnel = ngrok.connect(
                self.WEBHOOK_PORT,
                bind_tls=True,
                proto="http"
            )
            time.sleep(1.5)
            return self.ngrok_tunnel.public_url
        except Exception as e:
            raise Exception(f"Failed to start ngrok: {e}")

    def _kill_existing_ngrok(self):
        """Kill t■t c■ ngrok processes"""
        for proc in psutil.process_iter(['pid', 'name']):
            try:
                if proc.info['name'] and "ngrok" in proc.info['name'].lower():
                    proc.kill()
            except:
                pass

    def _kill_port_process(self, port: int):
        """Kill process ■ang dùng port"""
        for proc in psutil.process_iter(['pid', 'name']):
            try:
                for conn in proc.net_connections():
                    if conn.laddr.port == port:
                        print(f"[WebhookService] Killing process on port {port}")
                        proc.kill()
                        time.sleep(2)
                        break
            except:
                pass

    def _start_fastapi_server(self):
        """Kh■i ■■ng FastAPI server trong subprocess"""
        import subprocess
        cmd = [
            "uvicorn",
            "api.webhook_app:app",
            "--host", "0.0.0.0",
            "--port", str(self.WEBHOOK_PORT)
        ]
        self.server_process = subprocess.Popen(cmd)

    def _create_subscription(self) -> Optional[str]:
        """T■o Microsoft Graph subscription"""
        token = get_token()
        headers = {
            "Authorization": f"Bearer {token}",
            "Content-Type": "application/json"
        }

        notification_url = f"{self.public_url}/webhook/notifications"
        exp = (datetime.now(timezone.utc) + timedelta(days=3)).isoformat()

        payload = {
            "changeType": "created",
            "notificationUrl": notification_url,
            "resource": "me/mailfolders('inbox')/messages",
            "expirationDateTime": exp,
            "clientState": "webhook_secret_state"
        }

        try:
            resp = requests.post(
                f"{self.GRAPH_URL}/subscriptions",
                headers=headers,
                json=payload,
                timeout=30
            )

            if resp.status_code == 201:
                data = resp.json()
                sub_id = data.get("id")

                # Save subscription to Redis
                session_manager.redis.save_subscription(data)

                return sub_id

            print(f"[WebhookService] Subscription creation failed: {resp.text}")
            return None

        except Exception as e:
            print(f"[WebhookService] Subscription error: {e}")
            return None

    def _delete_subscription(self):
        """Xóa subscription"""
        if not self.subscription_id:
            return

        token = get_token()
        headers = {"Authorization": f"Bearer {token}"}

        try:
            requests.delete(
                f"{self.GRAPH_URL}/subscriptions/{self.subscription_id}",
                headers=headers,
                timeout=10
            )
            print(f"[WebhookService] Subscription deleted")
        except Exception as e:
            print(f"[WebhookService] Delete subscription error: {e}")

    def _renew_subscription(self) -> bool:
        """Renew subscription"""
        if not self.subscription_id:
            return False

        token = get_token()
        headers = {
            "Authorization": f"Bearer {token}",
            "Content-Type": "application/json"
        }

        new_exp = (datetime.now(timezone.utc) + timedelta(days=3)).isoformat()
        payload = {"expirationDateTime": new_exp}

        try:
            resp = requests.patch(
                f"{self.GRAPH_URL}/subscriptions/{self.subscription_id}",
                headers=headers,
                json=payload,
                timeout=10
            )

            if resp.status_code == 200:
                print(f"[WebhookService] Subscription renewed until {new_exp}")
                return True

            return False
        except Exception as e:
            print(f"[WebhookService] Renew error: {e}")
            return False

    def _start_renewal_watcher(self):
        """Kh■i ■■ng watcher t■ ■■ng renew subscription"""
        def renewal_loop():
            check_interval = 300  # 5 phút
            threshold_hours = 1

            while self.active:
                try:
                    time.sleep(check_interval)

                    # Get subscription status
                    token = get_token()
                    headers = {"Authorization": f"Bearer {token}"}

                    resp = requests.get(
                        f"{self.GRAPH_URL}/subscriptions/{self.subscription_id}",
                        headers=headers,
                        timeout=10
                    )

                    if resp.status_code != 200:
                        print(f"[WebhookService] Subscription not found, recreating...")
                        self.subscription_id = self._create_subscription()
                        continue

                    sub = resp.json()
                    exp_str = sub.get("expirationDateTime")
                    exp_dt = datetime.fromisoformat(exp_str.replace("Z", "+00:00"))
                    remaining = exp_dt - datetime.now(timezone.utc)
                    hours_left = remaining.total_seconds() / 3600

                    # Renew if needed
                    if hours_left < threshold_hours:
                        print(f"[WebhookService] Renewing (only {hours_left:.1f}h left)")
                        self._renew_subscription()

                except Exception as e:
                    print(f"[WebhookService] Renewal watcher error: {e}")

        thread = threading.Thread(target=renewal_loop, daemon=True)
        thread.start()

    def get_status(self) -> Dict:
        """L■y tr■ng thái webhook service"""
        return {
            "active": self.active,
            "public_url": self.public_url,
            "subscription_id": self.subscription_id,
            "error_count": self.error_count,
            "port": self.WEBHOOK_PORT
        }

# Singleton instance
webhook_service = WebhookService()
```