

## Source: C:\Users\Admin\Desktop\CMC\ms1\_email ngestion\core\session\_manager.py

```
"""
Session Manager - Core component for managing work sessions
Handles state transitions between polling and webhook modes
"""

import json
import os
import threading
from datetime import datetime, timezone
from enum import Enum
from typing import Optional, Dict, Set
from dataclasses import dataclass, asdict
from concurrent_storage.redis_manager import get_redis_storage

class SessionState(Enum):
    """Trạng thái phiên làm việc"""
    IDLE = "idle"
    POLLING_ACTIVE = "polling_active"
    WEBHOOK_ACTIVE = "webhook_active"
    BOTH_ACTIVE = "both_active"
    TERMINATED = "terminated"
    ERROR = "error"

class TriggerMode(Enum):
    """Cách trigger polling"""
    MANUAL = "manual"
    SCHEDULED = "scheduled"
    FALLBACK = "fallback"

@dataclass
class SessionConfig:
    """Cấu hình phiên làm việc"""
    session_id: str
    start_time: str
    polling_interval: int = 300 # 5 phút
    webhook_enabled: bool = True
    polling_mode: str = TriggerMode.SCHEDULED.value
    max_polling_errors: int = 3
    max_webhook_errors: int = 5

class SessionManager:
    """Quản lý phiên làm việc với Redis backend"""

    def __init__(self):
        self.redis = get_redis_storage()
        self.config: Optional[SessionConfig] = None
        self.state = SessionState.IDLE
        self._load_state()

    def start_session(self, config: SessionConfig) -> bool:
        """Khởi động phiên làm việc mới"""
        current_state = self.redis.get_session_state()
        if current_state and current_state.get("state") not in ["idle", "terminated"]:
            print(f"[SessionManager] Cannot start: current state is {current_state.get('state')}")
            return False

        self.config = config

        # Xác định state dựa trên config
        if config.webhook_enabled:
            self.state = SessionState.BOTH_ACTIVE
        else:
            self.state = SessionState.POLLING_ACTIVE

        session_data = {
            "session_id": config.session_id,
            "state": self.state.value,
            "start_time": config.start_time,
            "polling_interval": config.polling_interval,
            "webhook_enabled": config.webhook_enabled,
            "polling_mode": config.polling_mode,
            "polling_errors": 0,
            "webhook_errors": 0,
```

```

        "processed_count": 0,
        "pending_count": 0,
        "timestamp": datetime.now(timezone.utc).isoformat()
    }

    self.redis.set_session_state(session_data)
    print(f"[SessionManager] Session {config.session_id} started")

    # Hiển thị mode phù hợp
    if config.webhook_enabled:
        print(f"[SessionManager] Mode: BOTH_ACTIVE (Polling + Webhook)")
    else:
        print(f"[SessionManager] Mode: POLLING_ACTIVE (Polling only)")

    return True

def complete_initial_polling(self) -> bool:
    """Hoàn thành giai đoạn polling ban đầu"""
    current_state = self.redis.get_session_state()
    if not current_state or current_state.get("state") != SessionState.BOTH_ACTIVE.value:
        print(f"[SessionManager] Invalid transition from {current_state.get('state')} if current state is None")
        return False

    self.state = SessionState.WEBHOOK_ACTIVE
    self.redis.update_session_field("state", self.state.value)
    self.redis.update_session_field("timestamp", datetime.now(timezone.utc).isoformat())

    print(f"[SessionManager] Initial polling completed")
    print(f"[SessionManager] Mode: WEBHOOK_ACTIVE (Webhook only)")
    return True

def activate_fallback_polling(self, reason: str) -> bool:
    """Kích hoạt polling dự phòng khi webhook lỗi"""
    current_state = self.redis.get_session_state()
    if not current_state:
        return False

    if current_state.get("state") == SessionState.WEBHOOK_ACTIVE.value:
        self.state = SessionState.BOTH_ACTIVE

        self.redis.update_session_field("state", self.state.value)
        webhook_errors = self.redis.increment_session_counter("webhook_errors")
        self.redis.update_session_field("fallback_reason", reason)
        self.redis.update_session_field("timestamp", datetime.now(timezone.utc).isoformat())

        print(f"[SessionManager] FALBACK activated: {reason}")
        print(f"[SessionManager] Webhook errors: {webhook_errors}")
        return True

    return False

def restore_webhook_only(self) -> bool:
    """Khôi phục chế độ chỉ webhook sau khi sập lỗi"""
    current_state = self.redis.get_session_state()
    if not current_state:
        return False

    if current_state.get("state") == SessionState.BOTH_ACTIVE.value:
        self.state = SessionState.WEBHOOK_ACTIVE

        self.redis.update_session_field("state", self.state.value)
        self.redis.update_session_field("webhook_errors", "0")
        self.redis.update_session_field("timestamp", datetime.now(timezone.utc).isoformat())

        print(f"[SessionManager] Webhook restored, polling deactivated")
        return True

    return False

def register_processed_email(self, email_id: str) -> bool:
    """Đăng ký email đã xử lý"""
    is_new = self.redis.mark_email_processed(email_id)

    if is_new:
        self.redis.remove_pending(email_id)
        self.redis.increment_session_counter("processed_count")
        self.redis.increment_metric("emails_processed")

```

```

        self.redis.increment_counter("total_processed")

    return is_new

def register_pending_email(self, email_id: str):
    """Thêm ký email đang chờ xử lý"""
    if not self.redis.is_email_processed(email_id):
        self.redis.add_pending_email(email_id)
        pending_count = self.redis.get_pending_count()
        self.redis.update_session_field("pending_count", pending_count)

def register_failed_email(self, email_id: str, error: str):
    """Thêm ký email xử lý thất bại"""
    self.redis.move_to_failed(email_id, error)
    self.redis.increment_session_counter("failed_count")
    self.redis.increment_metric("emails_failed")

def increment_polling_errors(self) -> int:
    """Tăng số lượng lỗi polling"""
    return self.redis.increment_session_counter("polling_errors")

def increment_webhook_errors(self) -> int:
    """Tăng số lượng lỗi webhook"""
    return self.redis.increment_session_counter("webhook_errors")

def terminate_session(self, reason: str = "user_requested"):
    """Kết thúc phiên làm việc"""
    current_state = self.redis.get_session_state()
    if not current_state:
        print("[SessionManager] No active session to terminate")
        return

    print(f"[SessionManager] Terminating session: {reason}")

    processed = self.redis.get_processed_count()
    pending = self.redis.get_pending_count()
    failed = self.redis.get_failed_count()

    print(f"[SessionManager] Processed emails: {processed}")
    print(f"[SessionManager] Pending emails: {pending}")
    print(f"[SessionManager] Failed emails: {failed}")

    self.state = SessionState.TERMINATED
    self.redis.update_session_field("state", self.state.value)
    self.redis.update_session_field("end_time", datetime.now(timezone.utc).isoformat())
    self.redis.update_session_field("termination_reason", reason)

    final_state = self.redis.get_session_state()
    self.redis.save_session_history(final_state)

def get_session_status(self) -> Dict:
    """Lấy trạng thái phiên hiện tại"""
    session_data = self.redis.get_session_state()

    if not session_data:
        return {
            "session_id": None,
            "state": SessionState.IDLE.value,
            "processed_count": 0,
            "pending_count": 0,
            "failed_count": 0,
            "polling_errors": 0,
            "webhook_errors": 0,
            "timestamp": datetime.now(timezone.utc).isoformat()
        }

    processed_count = self.redis.get_processed_count()
    pending_count = self.redis.get_pending_count()
    failed_count = self.redis.get_failed_count()

    session_data["processed_count"] = processed_count
    session_data["pending_count"] = pending_count
    session_data["failed_count"] = failed_count

    for key in ["polling_errors", "webhook_errors", "polling_interval"]:
        if key in session_data:
            session_data[key] = int(session_data.get(key, 0))

```

```
        return session_data

    def is_email_processed(self, email_id: str) -> bool:
        """Kiểm tra email đã được xử lý chưa"""
        return self.redis.is_email_processed(email_id)

    def get_metrics(self) -> Dict:
        """Lấy metrics tổng hợp"""
        today_metrics = self.redis.get_metrics()
        total_processed = self.redis.get_counter("total_processed")
        session_status = self.get_session_status()

        return {
            "today": today_metrics,
            "lifetime": {
                "total_processed": total_processed
            },
            "session": session_status
        }

    def _load_state(self):
        """Load state từ Redis khi khởi động"""
        session_data = self.redis.get_session_state()
        if session_data:
            state_str = session_data.get("state", "idle")
            try:
                self.state = SessionState(state_str)
            except ValueError:
                self.state = SessionState.IDLE

# Singleton instance
session_manager = SessionManager()
```