```python
import discord
from discord.ext import commands, tasks
import aiohttp
import asyncio
import json
import qrcode
import io
import os
from datetime import datetime
import logging
import hashlib
import time
from web3 import Web3
import requests

# Configuration
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Tokens et configuration
DISCORD_TOKEN = os.getenv('DISCORD_TOKEN')
ANTHROPIC_API_KEY = os.getenv('ANTHROPIC_API_KEY')
PINATA_API_KEY = os.getenv('PINATA_API_KEY')
PINATA_SECRET = os.getenv('PINATA_SECRET')
WEB3_PROVIDER_URL = os.getenv('WEB3_PROVIDER_URL')  # Sepolia RPC
CONTRACT_ADDRESS = os.getenv('CONTRACT_ADDRESS')
PRIVATE_KEY = os.getenv('PRIVATE_KEY')

intents = discord.Intents.default()
intents.message_content = True
intents.guilds = True

bot = commands.Bot(command_prefix='!md ', intents=intents)

class MonsterDogIntegratedCore:
    def __init__(self):
        self.sessions = {}
        self.logs = []
        self.active_portals = []
        self.nft_mints = []
        self.ipfs_pins = []
        self.web3 = None
        self.contract = None

        # Initialize Web3 if configured
        if WEB3_PROVIDER_URL and CONTRACT_ADDRESS:
            self.init_web3()

    def init_web3(self):
        """Initialize Web3 connection and contract"""
        try:
            self.web3 = Web3(Web3.HTTPProvider(WEB3_PROVIDER_URL))

            # Load contract ABI (simplified for demo)
            contract_abi = [
                {
                    "inputs": [],
                    "name": "getCurrentTokenId",
                    "outputs": [{"internalType": "uint256", "name": "", "type": "uint256"}],
                    "stateMutability": "view",
                    "type": "function"
                },
                {
                    "inputs": [{"internalType": "address", "name": "owner", "type":
"address"}],
                    "name": "getTokensByOwner",
                    "outputs": [{"internalType": "uint256[]", "name": "", "type":
"uint256[]"}],
                    "stateMutability": "view",
                    "type": "function"
                }
            ]

            self.contract = self.web3.eth.contract(
                address=Web3.toChecksumAddress(CONTRACT_ADDRESS),
                abi=contract_abi
            )
            logger.info("Web3 initialized successfully")
        except Exception as e:
            logger.error(f"Web3 initialization failed: {e}")

    def log_action(self, user_id, action, details, success=True):
        """Log toutes les actions du bot"""
        log_entry = {
            'timestamp': datetime.now().isoformat(),
            'user_id': user_id,
            'action': action,
            'details': details,
            'success': success,
            'session_id': f"md_{int(time.time())}"
        }
        self.logs.append(log_entry)

        # Garde seulement les 1000 derniers logs
        if len(self.logs) > 1000:
            self.logs = self.logs[-1000:]

# Instance globale
core = MonsterDogIntegratedCore()

@bot.event
async def on_ready():
    print(f'🎯 {bot.user} online - MONSTERDOG INTEGRATED CORE ACTIVE!')
    print(f'🔗 Connected to {len(bot.guilds)} servers')
    print(f'🌐 Web3 Status: {"✅ Connected" if core.web3 else "❌ Not configured"}')

    # Start background tasks
    if not portal_status_monitor.is_running():
        portal_status_monitor.start()

@tasks.loop(minutes=10)
async def portal_status_monitor():
    """Monitor portal status et update metrics"""
    try:
        # Check active portals
        for portal in core.active_portals:
            # Simulate portal health check
            portal['last_check'] = datetime.now().isoformat()
            portal['status'] = 'active' if time.time() - portal.get('created', 0) < 3600 else
'inactive'

        # Log monitoring
        core.log_action(0, 'portal_monitor', f'Checked {len(core.active_portals)} portals')

    except Exception as e:
        logger.error(f"Portal monitoring error: {e}")

@bot.command(name='create_portal')
async def create_xr_portal(ctx, mode="FRACTAL"):
    """Crée un nouveau portail XR avec déploiement IPFS"""
    await ctx.trigger_typing()

    try:
        # Generate session data
        session_data = {
            "session": f"MONSTERDOG_XR_SESSION_{int(time.time())}",
            "timestamp": datetime.now().isoformat(),
            "creator": str(ctx.author.id),
            "guild": str(ctx.guild.id) if
```