```
"""
ERES Institute PlayNAC-KERNEL Framework
Empirical Realtime Education System with LOGOS Smart-City Integration

Based on Joseph A. Sprute's (ERES Maestro) New Age Cybernetics
Repository: https://github.com/ERES-Institute-for-New-Age-Cybernetics/Proof-of-Work_MD

Core Frameworks:
- PERC (Personal Ecologic Ratings Codex)
- BERC (Bio-Ecologic Ratings Codex)
- JERC (Judicial Ecologic Ratings Codex)
- LOGOS (Locational, Organizational, Governance, Operational, Societal)
- UBIMIA (Universal Basic Income + Merit-based Incentives & Awards)
- GraceChain (Blockchain for ethical verification)
- Meritcoin (Digital unit of ethical value)
- EarnedPath (Merit progression system)
- NBERS (National Bio-Ecologic Resource Score)
- REACI (Resonant-Ecologic Adaptive Civic Infrastructure)
- SROC (Smart Registered Offset Contracts)

License: CARE Commons Attribution License v2.1 (CCAL)
"""

from dataclasses import dataclass, field
from typing import Dict, List, Optional, Set, Tuple
from enum import Enum
from datetime import datetime
import json
from collections import defaultdict

#
================================================================================
====
# CORE ENUMERATIONS
#
================================================================================
====

class UserGroupType(Enum):
    """LOGOS Smart-City User Group Classifications"""
    RESIDENTIAL = "residential"
    COMMERCIAL = "commercial"
    EDUCATIONAL = "educational"
    RECREATIONAL = "recreational"
```

```
    INDUSTRIAL = "industrial"
    CULTURAL = "cultural"
    GOVERNANCE = "governance"

class MeritLevel(Enum):
    """EarnedPath Merit Progression Levels"""
    SEEKER = 1      # Beginning journey
    LEARNER = 2     # Active learning
    CONTRIBUTOR = 3 # Contributing to community
    MENTOR = 4      # Guiding others
    MAESTRO = 5     # Master level (Joseph A. Sprute level)

class ResourceType(Enum):
    """Core Resources per NAC Principles"""
    CLEAN_WATER = "clean_water"
    FOOD = "food"
    SHELTER = "shelter"
    WORK = "work"
    LOVE = "love"  # Community connection
    ENERGY = "energy"  # Renewable energy access

class GovernanceLicense(Enum):
    """LOGOS Governance License Types"""
    CIL = "community_implementation_license"  # Community/district level
    MGL = "municipal_governance_license"      # City-wide deployment

#
================================================================================
====
# RATING SYSTEMS (PERC, BERC, JERC)
#
================================================================================
====

@dataclass
class PERCScore:
    """
    Personal Ecologic Ratings Codex
    Individual's ecological impact and contribution
    """
    user_id: str
    carbon_footprint: float = 0.0
    renewable_energy_use: float = 0.0
    waste_reduction: float = 0.0
```

```
    conservation_actions: int = 0
    education_hours: float = 0.0

    def calculate_score(self) -> float:
        """Calculate overall PERC score (0-100)"""
        score = (
            (100 - min(100, self.carbon_footprint)) * 0.3 +
            self.renewable_energy_use * 0.3 +
            self.waste_reduction * 0.2 +
            min(100, self.conservation_actions * 2) * 0.1 +
            min(100, self.education_hours) * 0.1
        )
        return min(100, score)

@dataclass
class BERCScore:
    """
    Bio-Ecologic Ratings Codex
    Community/location ecological health
    """
    location_id: str
    air_quality_index: float = 50.0  # 0-100, higher is better
    water_quality_index: float = 50.0
    biodiversity_score: float = 50.0
    renewable_energy_ratio: float = 0.0  # % of energy from renewables
    green_space_ratio: float = 0.0  # % area as green space

    def calculate_nbers(self) -> float:
        """
        Calculate National Bio-Ecologic Resource Score (NBERS)
        Harmony between human activity, resources, and ecology
        """
        nbers = (
            self.air_quality_index * 0.25 +
            self.water_quality_index * 0.25 +
            self.biodiversity_score * 0.20 +
            self.renewable_energy_ratio * 0.20 +
            self.green_space_ratio * 0.10
        )
        return nbers

@dataclass
class JERCScore:
    """
```

```
    Judicial Ecologic Ratings Codex
    Legal/justice system alignment with ecological ethics
    """
    jurisdiction_id: str
    environmental_protections: int = 0  # Number of active protections
    enforcement_rate: float = 0.0  # 0-1.0
    restorative_justice_programs: int = 0
    ecological_crime_penalties: float = 0.0

    def calculate_score(self) -> float:
        """Calculate JERC score"""
        return min(100, (
            self.environmental_protections * 2 +
            self.enforcement_rate * 30 +
            self.restorative_justice_programs * 5 +
            self.ecological_crime_penalties * 0.5
        ))

#
================================================================================
====
# BLOCKCHAIN & CURRENCY SYSTEMS
#
================================================================================
====

@dataclass
class GraceChainBlock:
    """
    GraceChain: Blockchain for ethical verification and merit tracking
    Immutable, transparent ledger for all contributions
    """
    block_id: str
    timestamp: datetime
    transactions: List[Dict] = field(default_factory=list)
    previous_hash: str = ""
    hash: str = ""

    def add_transaction(self, transaction: Dict):
        """Add verified transaction to block"""
        self.transactions.append({
            **transaction,
            "timestamp": datetime.now().isoformat(),
            "verified": True
```

```
        })

@dataclass
class MeritcoinAccount:
    """
    Meritcoin: Digital unit of ethical and empirical value
    Minted only via GraceChain-verified activity aligned with NAC principles
    """
    user_id: str
    balance: float = 0.0
    total_earned: float = 0.0
    total_spent: float = 0.0
    transactions: List[Dict] = field(default_factory=list)

    def mint_meritcoin(self, amount: float, reason: str,
                       verification_hash: str) -> bool:
        """
        Mint new Meritcoin for verified contributions
        Tied to personal wellbeing, pain mitigation, and group benefit
        """
        self.balance += amount
        self.total_earned += amount
        self.transactions.append({
            "type": "mint",
            "amount": amount,
            "reason": reason,
            "verification": verification_hash,
            "timestamp": datetime.now().isoformat()
        })
        return True

    def transfer(self, to_user: str, amount: float, reason: str) -> bool:
        """Transfer Meritcoin to another user"""
        if self.balance >= amount:
            self.balance -= amount
            self.total_spent += amount
            self.transactions.append({
                "type": "transfer",
                "to": to_user,
                "amount": amount,
                "reason": reason,
                "timestamp": datetime.now().isoformat()
            })
            return True
```

```
        return False

@dataclass
class UBIMIAAccount:
    """
    UBIMIA: Universal Basic Income + Merit-based Incentives & Awards
    Combines guaranteed baseline with performance rewards
    """
    user_id: str
    ubi_balance: float = 100.0  # Universal baseline
    merit_balance: float = 0.0  # Earned through contributions
    awards: List[Dict] = field(default_factory=list)

    def monthly_ubi_distribution(self, amount: float = 100.0):
        """Distribute monthly UBI"""
        self.ubi_balance += amount

    def award_merit(self, amount: float, achievement: str):
        """Award merit-based incentive"""
        self.merit_balance += amount
        self.awards.append({
            "amount": amount,
            "achievement": achievement,
            "timestamp": datetime.now().isoformat()
        })

    def total_balance(self) -> float:
        """Total purchasing power"""
        return self.ubi_balance + self.merit_balance

#
================================================================================
====
# SMART CITY INFRASTRUCTURE (LOGOS)
#
================================================================================
====

@dataclass
class LOGOSLocation:
    """
    Location: Evaluated and adaptive physical setting
    Assessed using NBERS for ecological harmony
    """
```

```
    location_id: str
    name: str
    coordinates: Tuple[float, float]
    berc_score: BERCScore
    governance_license: GovernanceLicense

    def get_nbers(self) -> float:
        """Get National Bio-Ecologic Resource Score"""
        return self.berc_score.calculate_nbers()

    def is_suitable_for_community(self, min_nbers: float = 60.0) -> bool:
        """Check if location meets NBERS threshold"""
        return self.get_nbers() >= min_nbers

@dataclass
class REACISystem:
    """
    REACI: Resonant-Ecologic Adaptive Civic Infrastructure
    Dynamic urban planning that adjusts to real-time data
    Supports non-punitive migration
    """
    city_id: str
    infrastructure_data: Dict[str, float] = field(default_factory=dict)
    adaptation_history: List[Dict] = field(default_factory=list)
    migration_support: bool = True

    def adapt_infrastructure(self, metric: str, new_value: float,
                             reason: str):
        """Adjust infrastructure based on ecological/social data"""
        old_value = self.infrastructure_data.get(metric, 0.0)
        self.infrastructure_data[metric] = new_value
        self.adaptation_history.append({
            "metric": metric,
            "old_value": old_value,
            "new_value": new_value,
            "reason": reason,
            "timestamp": datetime.now().isoformat()
        })

    def support_migration(self, population_change: int):
        """Handle population movements without penalties"""
        if self.migration_support:
            self.adapt_infrastructure(
                "population",
```

```
            self.infrastructure_data.get("population", 0) + population_change,
            "Non-punitive migration support"
        )

@dataclass
class SROCContract:
    """
    SROC: Smart Registered Offset Contracts
    Blockchain-based environmental credit agreements
    Backed by actual renewable energy generation data
    """
    contract_id: str
    issuer: str
    offset_type: str  # carbon_capture, renewable_energy, reforestation
    credits: float
    verification_data: Dict = field(default_factory=dict)
    blockchain_record: str = ""

    def verify_offset(self, actual_data: Dict) -> bool:
        """Verify offset against actual performance data"""
        self.verification_data = actual_data
        # Simplified verification - real system would validate against BERC
        return actual_data.get("verified", False)

    def trade_credits(self, buyer: str, amount: float) -> bool:
        """Trade offset credits"""
        if amount <= self.credits:
            self.credits -= amount
            return True
        return False

@dataclass
class SentientEnergyGrid:
    """
    Sentient Energy Grid: Self-monitoring renewable energy system
    Uses GSSG (Green Solar Sand Glass) building materials
    Real-time efficiency reporting
    """
    grid_id: str
    total_capacity: float
    current_output: float = 0.0
    efficiency: float = 0.85
    renewable_sources: Dict[str, float] = field(default_factory=dict)
```

```
    def adjust_output(self):
        """Self-adjust based on demand and conditions"""
        self.current_output = self.total_capacity * self.efficiency

    def generate_sroc(self, period_days: int) -> SROCContract:
        """Generate SROC based on actual production"""
        energy_produced = self.current_output * period_days * 24
        return SROCContract(
            contract_id=f"{self.grid_id}_sroc_{datetime.now().timestamp()}",
            issuer=self.grid_id,
            offset_type="renewable_energy",
            credits=energy_produced / 1000,  # Convert to carbon credits
            verification_data={"energy_kwh": energy_produced, "verified": True}
        )

#
================================================================================
====
# PLAYNAC & EARNEDPATH
#
================================================================================
====

@dataclass
class EarnedPathProfile:
    """
    EarnedPath: Role-based contribution-tracked membership
    Progress tracked via secure GraceChain ledger
    """
    user_id: str
    merit_level: MeritLevel = MeritLevel.SEEKER
    total_contributions: int = 0
    skills: Set[str] = field(default_factory=set)
    certifications: List[Dict] = field(default_factory=list)
    learning_paths_completed: int = 0

    def add_contribution(self, contribution_type: str):
        """Record contribution and potentially level up"""
        self.total_contributions += 1
        if self.should_level_up():
            self.level_up()

    def should_level_up(self) -> bool:
        """Check if user qualifies for next level"""
```

```
        thresholds = {
            MeritLevel.SEEKER: 10,
            MeritLevel.LEARNER: 50,
            MeritLevel.CONTRIBUTOR: 150,
            MeritLevel.MENTOR: 500
        }
        threshold = thresholds.get(self.merit_level, float('inf'))
        return self.total_contributions >= threshold

    def level_up(self):
        """Advance to next merit level"""
        levels = list(MeritLevel)
        current_index = levels.index(self.merit_level)
        if current_index < len(levels) - 1:
            self.merit_level = levels[current_index + 1]

    def add_skill(self, skill: str, blockchain_cert: str):
        """Add verified skill with blockchain certification"""
        self.skills.add(skill)
        self.certifications.append({
            "skill": skill,
            "verification": blockchain_cert,
            "timestamp": datetime.now().isoformat()
        })

@dataclass
class PlayNACActivity:
    """
    PlayNAC: Game Theory Application for New Age Cybernetics
    Educational activities that generate real value
    """
    activity_id: str
    name: str
    description: str
    participants: Set[str] = field(default_factory=set)
    learning_objectives: List[str] = field(default_factory=list)
    meritcoin_reward: float = 0.0
    perc_impact: float = 0.0  # PERC score improvement
    collaborative: bool = True

    def complete_activity(self, user_id: str, performance: float,
                          gracechain: GraceChainBlock) -> Tuple[float, float]:
        """Complete PlayNAC activity, record on GraceChain"""
        self.participants.add(user_id)
```

```
        base_reward = self.meritcoin_reward * performance

        # Collaborative bonus
        if self.collaborative and len(self.participants) > 1:
            collab_bonus = min(0.5, len(self.participants) * 0.1)
            base_reward *= (1 + collab_bonus)

        # Record on GraceChain
        gracechain.add_transaction({
            "type": "playnac_completion",
            "activity_id": self.activity_id,
            "user_id": user_id,
            "meritcoin_earned": base_reward,
            "perc_improvement": self.perc_impact * performance
        })

        return base_reward, self.perc_impact * performance

#
================================================================================
====
# MAIN ERES SYSTEM WITH FULL NAC INTEGRATION
#
================================================================================
====

class ERESPlayNACSystem:
    """
    Complete ERES Institute System with LOGOS Smart-City Integration

    Integrates:
    - PlayNAC-KERNEL
    - PERC/BERC/JERC ratings
    - GraceChain blockchain
    - Meritcoin & UBIMIA economy
    - LOGOS smart city framework
    - REACI adaptive infrastructure
    - SROC environmental credits
    - EarnedPath progression
    """

    def __init__(self):
        # User systems
        self.earned_paths: Dict[str, EarnedPathProfile] = {}
```

```
        self.perc_scores: Dict[str, PERCScore] = {}
        self.meritcoin_accounts: Dict[str, MeritcoinAccount] = {}
        self.ubimia_accounts: Dict[str, UBIMIAAccount] = {}

        # Smart city systems
        self.locations: Dict[str, LOGOSLocation] = {}
        self.berc_scores: Dict[str, BERCScore] = {}
        self.jerc_scores: Dict[str, JERCScore] = {}
        self.reaci_systems: Dict[str, REACISystem] = {}
        self.energy_grids: Dict[str, SentientEnergyGrid] = {}

        # Blockchain & activities
        self.gracechain: List[GraceChainBlock] = []
        self.activities: Dict[str, PlayNACActivity] = {}
        self.sroc_contracts: Dict[str, SROCContract] = {}

        # System metadata
        self.system_timestamp = datetime.now()
        self.governance_license = GovernanceLicense.CIL

        # Initialize genesis block
        self._create_genesis_block()

    def _create_genesis_block(self):
        """Create GraceChain genesis block"""
        genesis = GraceChainBlock(
            block_id="genesis",
            timestamp=self.system_timestamp,
            previous_hash="0",
            hash="genesis_hash"
        )
        genesis.add_transaction({
            "type": "system_initialization",
            "message": "ERES Institute PlayNAC-KERNEL System Genesis",
            "author": "Joseph A. Sprute, ERES Maestro",
            "license": "CARE Commons Attribution License v2.1 (CCAL)"
        })
        self.gracechain.append(genesis)

    def register_user(self, user_id: str) -> Dict[str, any]:
        """Register user in complete ERES system"""
        # Create all user accounts
        self.earned_paths[user_id] = EarnedPathProfile(user_id)
        self.perc_scores[user_id] = PERCScore(user_id)
```

```
    self.meritcoin_accounts[user_id] = MeritcoinAccount(user_id)
    self.ubimia_accounts[user_id] = UBIMIAAccount(user_id)

    # Record on GraceChain
    current_block = self.gracechain[-1]
    current_block.add_transaction({
        "type": "user_registration",
        "user_id": user_id,
        "initial_merit_level": MeritLevel.SEEKER.name
    })

    print(f"✓ User {user_id} registered in ERES PlayNAC-KERNEL system")

    return {
        "user_id": user_id,
        "merit_level": MeritLevel.SEEKER.name,
        "meritcoin_balance": 0.0,
        "ubimia_balance": 100.0,
        "perc_score": 0.0
    }

def create_smart_city_location(self, location_id: str, name: str,
                               coords: Tuple[float, float],
                               initial_berc: Dict) -> LOGOSLocation:
    """Create LOGOS smart city location with NBERS"""
    berc = BERCScore(
        location_id=location_id,
        **initial_berc
    )
    self.berc_scores[location_id] = berc

    location = LOGOSLocation(
        location_id=location_id,
        name=name,
        coordinates=coords,
        berc_score=berc,
        governance_license=self.governance_license
    )
    self.locations[location_id] = location

    # Create REACI system for location
    self.reaci_systems[location_id] = REACISystem(city_id=location_id)

    nbers = location.get_nbers()
```

```
    print(f"✓ Smart City '{name}' created - NBERS: {nbers:.2f}")

    return location

def deploy_energy_grid(self, grid_id: str, capacity: float) -> SentientEnergyGrid:
    """Deploy Sentient Energy Grid with GSSG"""
    grid = SentientEnergyGrid(
        grid_id=grid_id,
        total_capacity=capacity,
        renewable_sources={"solar": 0.6, "wind": 0.3, "hydro": 0.1}
    )
    grid.adjust_output()
    self.energy_grids[grid_id] = grid

    print(f"✓ Sentient Energy Grid '{grid_id}' deployed - Capacity: {capacity}kW")
    return grid

def participate_in_playnac(self, user_id: str, activity_id: str,
                           performance: float) -> Dict:
    """User participates in PlayNAC activity"""
    if user_id not in self.earned_paths:
        raise ValueError(f"User {user_id} not registered")
    if activity_id not in self.activities:
        raise ValueError(f"Activity {activity_id} does not exist")

    activity = self.activities[activity_id]
    current_block = self.gracechain[-1]

    # Complete activity and record on GraceChain
    meritcoin_earned, perc_improvement = activity.complete_activity(
        user_id, performance, current_block
    )

    # Update all user systems
    self.meritcoin_accounts[user_id].mint_meritcoin(
        meritcoin_earned,
        f"PlayNAC: {activity.name}",
        current_block.hash
    )

    self.perc_scores[user_id].conservation_actions += 1

    self.earned_paths[user_id].add_contribution("playnac_activity")
```

```
    # UBIMIA merit award
    self.ubimia_accounts[user_id].award_merit(
        meritcoin_earned * 0.5,
        f"PlayNAC Achievement: {activity.name}"
    )

    print(f"✓ {user_id} completed '{activity.name}'")
    print(f"  Meritcoin: +{meritcoin_earned:.2f}")
    print(f"  PERC Impact: +{perc_improvement:.2f}")

    return {
        "meritcoin_earned": meritcoin_earned,
        "perc_improvement": perc_improvement,
        "new_merit_level": self.earned_paths[user_id].merit_level.name
    }

def generate_sroc_from_grid(self, grid_id: str, period_days: int) -> SROCContract:
    """Generate SROC from energy grid performance"""
    if grid_id not in self.energy_grids:
        raise ValueError(f"Grid {grid_id} not found")

    grid = self.energy_grids[grid_id]
    sroc = grid.generate_sroc(period_days)
    self.sroc_contracts[sroc.contract_id] = sroc

    # Record on GraceChain
    current_block = self.gracechain[-1]
    current_block.add_transaction({
        "type": "sroc_generation",
        "contract_id": sroc.contract_id,
        "grid_id": grid_id,
        "credits": sroc.credits,
        "verification": sroc.verification_data
    })

    print(f"✓ SROC generated: {sroc.credits:.2f} credits")
    return sroc

def generate_comprehensive_report(self) -> Dict:
    """Generate full system report with all NAC metrics"""
    total_users = len(self.earned_paths)
    total_locations = len(self.locations)

    # Calculate aggregate metrics
```

```
total_meritcoin = sum(
    acc.balance for acc in self.meritcoin_accounts.values()
)
avg_perc = sum(
    score.calculate_score() for score in self.perc_scores.values()
) / max(1, len(self.perc_scores))

avg_nbers = sum(
    loc.get_nbers() for loc in self.locations.values()
) / max(1, len(self.locations))

total_sroc_credits = sum(
    contract.credits for contract in self.sroc_contracts.values()
)

merit_distribution = defaultdict(int)
for profile in self.earned_paths.values():
    merit_distribution[profile.merit_level.name] += 1

report = {
    "system": {
        "timestamp": datetime.now().isoformat(),
        "genesis_block": self.system_timestamp.isoformat(),
        "governance_license": self.governance_license.value,
        "gracechain_blocks": len(self.gracechain)
    },
    "users": {
        "total_registered": total_users,
        "merit_distribution": dict(merit_distribution),
        "average_perc_score": avg_perc
    },
    "economy": {
        "total_meritcoin_supply": total_meritcoin,
        "total_ubimia_distributed": sum(
            acc.total_balance() for acc in self.ubimia_accounts.values()
        )
    },
    "smart_cities": {
        "total_locations": total_locations,
        "average_nbers": avg_nbers,
        "energy_grids_deployed": len(self.energy_grids)
    },
    "environmental": {
        "total_sroc_credits": total_sroc_credits,
```

```
            "reaci_systems_active": len(self.reaci_systems)
        },
        "activities": {
            "playnac_activities": len(self.activities),
            "total_participants": sum(
                len(act.participants) for act in self.activities.values()
            )
        },
        "status": "operational",
        "version": "PlayNAC-KERNEL with LOGOS v2.0",
        "author": "Joseph A. Sprute, ERES Maestro",
        "license": "CARE Commons Attribution License v2.1 (CCAL)"
    }

    return report

#
================================================================================
====
# COMPREHENSIVE DEMONSTRATION
#
================================================================================
====

def demo_full_eres_system():
    """Demonstrate complete ERES PlayNAC-KERNEL with LOGOS integration"""

    print("=" * 80)
    print("ERES INSTITUTE - PlayNAC-KERNEL SYSTEM")
    print("Complete New Age Cybernetics Implementation")
    print("LOGOS Smart-City with PERC/BERC/JERC Integration")
    print()
    print("Author: Joseph A. Sprute, ERES Maestro")
    print("Repository: github.com/ERES-Institute-for-New-Age-Cybernetics")
    print("License: CARE Commons Attribution License v2.1 (CCAL)")
    print("=" * 80)
    print()

    # Initialize complete system
    system = ERESPlayNACSystem()

    # === USER REGISTRATION ===
    print(">>> PHASE 1: User Registration & EarnedPath Initialization")
    print("-" * 80)
```

```
users = ["alice", "bob", "carol"]
for user in users:
    system.register_user(user)
print()

# === SMART CITY CREATION ===
print(">>> PHASE 2: LOGOS Smart-City Location Creation")
print("-" * 80)
city = system.create_smart_city_location(
    "smart_city_001",
    "Bella Vista NAC Pilot Community",
    (36.4808, -94.2709),  # Northwest Arkansas coordinates
    {
        "air_quality_index": 75.0,
        "water_quality_index": 80.0,
        "biodiversity_score": 70.0,
        "renewable_energy_ratio": 45.0,
        "green_space_ratio": 25.0
    }
)
print(f"  NBERS Score: {city.get_nbers():.2f}/100")
print(f"  Governance: {city.governance_license.value}")
print()

# === ENERGY GRID DEPLOYMENT ===
print(">>> PHASE 3: Sentient Energy Grid Deployment")
print("-" * 80)
grid = system.deploy_energy_grid("grid_bella_vista", 5000.0)
print(f"  Current Output: {grid.current_output:.2f}kW")
print(f"  Efficiency: {grid.efficiency * 100:.1f}%")
print()

# === PLAYNAC ACTIVITIES ===
print(">>> PHASE 4: PlayNAC Activity Creation & Participation")
print("-" * 80)

# Create activities
system.activities["community_garden"] = PlayNACActivity(
    activity_id="community_garden",
    name="Community Permaculture Garden",
    description="Collaborative sustainable food production",
    learning_objectives=["Permaculture", "Water Conservation", "Biodiversity"],
    meritcoin_reward=50.0,
    perc_impact=10.0
```

```
    )

    system.activities["energy_conservation"] = PlayNACActivity(
        activity_i
```

REFERENCES:
https://claude.ai/public/artifacts/6b8af346-6965-4f80-b64b-74516bc3e29e
https://www.threads.com/@josephsprute/post/DPqx4cODcyM
https://claude.ai/public/artifacts/2466fa75-2c96-4aff-9ccc-4b3d97f50499