```
"""
Integrated Computational Pharmacognosy Engine
System-biology predictive engine translating botanical metadata into physiological outcomes
"""

import json
from typing import List, Dict, Optional, Tuple
from dataclasses import dataclass, asdict
from datetime import datetime
from enum import Enum


# ============================================================================
# IMPORT PHARMACOGNOSY DATABASE
# ============================================================================
# Assumes PharmacognosyDatabase class is available
# from pharmacognosy_database import PharmacognosyDatabase, GrowStyle, InterfaceTemp


# ============================================================================
# INTEGRATED ENGINE WITH LIVE DATABASE QUERIES
# ============================================================================

class IntegratedPharmacognosyEngine:
    """
    Production computational pharmacognosy engine
    Integrates recommendation algorithm with live database coefficients
    """

    def __init__(self, pharmacognosy_db, knowledge_base, precompute_engine=None):
        self.pharma_db = pharmacognosy_db
        self.kb = knowledge_base
        self.precompute = precompute_engine
        self.feedback_log = []

        # Learning parameters
        self.LEARNING_RATE = 0.1
        self.BASE_WEIGHTS = {
            'terpene': 0.5,
            'cannabinoid': 0.3,
            'flavonoid': 0.2
        }

    # ========================================================================
    # MAIN RANKING FUNCTION
    # ========================================================================

    def rank_products_integrated(self, user_profile: Dict,
                                 products: List[Dict]) -> List[Dict]:
        """
```

```
Main entry point: Integrated ranking with all logic blocks
"""
if not products or not user_profile:
    return []

# Get adaptive weights based on user history
weights = self._get_adaptive_weights(user_profile)

# Extract interface temperature for TRE logic
interface_temp = user_profile.get('interfaceTemp', 185)  # Default 185°C

ranked_results = []
all_scores = []

print(f"\n{'='*70}")
print(f"PROCESSING {len(products)} PRODUCTS")
print(f"Interface Temperature: {interface_temp}°C")
print(f"Adaptive Weights: T={weights['terpene']:.3f} | "
      f"C={weights['cannabinoid']:.3f} | F={weights['flavonoid']:.3f}")
print(f"{'='*70}\n")

# FIRST PASS: Calculate raw scores with all logic blocks
for product in products:
    scores = self._score_product_integrated(
        product, user_profile, weights, interface_temp
    )
    all_scores.append(scores['raw'])
    ranked_results.append(scores)

# CROSS-PRODUCT NORMALIZATION
if all_scores:
    max_score = max(all_scores)
    min_score = min(all_scores)
    score_range = max_score - min_score

    if score_range > 0:
        for result in ranked_results:
            result['matchScore'] = ((result['raw'] - min_score) / score_range) * 100
    else:
        for result in ranked_results:
            result['matchScore'] = 50.0

# Sort by normalized score
ranked_results.sort(key=lambda x: x['matchScore'], reverse=True)

# Apply synergy and safety overlays
for result in ranked_results:
    result['rationale'] = self._generate_medical_rationale(
        result, user_profile, weights
    )
```

```
        result['safetyWarnings'] = self._check_safety_overlays(
            result, user_profile, interface_temp
        )

    # Log for continuous learning
    self._log_recommendation(ranked_results, user_profile, weights)

    return ranked_results

# ============================================================================
# LOGIC BLOCK A: CULTIVATION POTENTIAL INDEX (CPI)
# ============================================================================

def _apply_cultivation_modifiers(self, compound_name: str, compound_class: str,
                                 grow_style: str, base_concentration: float) -> float:
    """
    Apply grow-style dependent multipliers to predict chemical biomass
    """
    modified_concentration = base_concentration

    if grow_style == "sun_grown":
        # UV-B Response: 1.40x for UV-responsive flavonoids
        if compound_class == "flavonoid" and compound_name in ["Quercetin", "Cannflavin A", "Cannflavin B"]:
            modified_concentration *= 1.40

    elif grow_style == "living_soil":
        # Rhizosphere Signaling: 1.25x for sesquiterpenes
        if compound_class == "terpene" and compound_name in ["β-Caryophyllene", "Humulene"]:
            modified_concentration *= 1.25
        # Also boost minor flavonoids
        elif compound_class == "flavonoid":
            modified_concentration *= 1.25

    elif grow_style == "drought_stress":
        # Metabolic Shift: 1.15x concentration across all
        modified_concentration *= 1.15
        # Note: Yield penalty handled separately in biomass calculations

    elif grow_style == "hydroponic":
        # High Potency: 1.10x for major cannabinoids
        if compound_class == "cannabinoid" and compound_name in ["THC", "CBD"]:
            modified_concentration *= 1.10
        # Low Diversity: 0.80x for flavonoids
        elif compound_class == "flavonoid":
            modified_concentration *= 0.80

    return modified_concentration

# ============================================================================
# LOGIC BLOCK B: THERMAL RELEASE EFFICIENCY (TRE)
```

```
# ========================================================================

def _calculate_thermal_availability(self, compound_name: str,
                                    compound_class: str,
                                    interface_temp: float) -> Tuple[float, str]:
    """
    Thermodynamic filter: Determine compound bioavailability at given temp
    Returns: (availability_multiplier, status_message)
    """
    # Query boiling point from database
    compound_data = None
    if compound_class == "terpene":
        compound_data = self.pharma_db.get_terpene(compound_name)
    elif compound_class == "cannabinoid":
        compound_data = self.pharma_db.get_cannabinoid(compound_name)
    elif compound_class == "flavonoid":
        compound_data = self.pharma_db.get_flavonoid(compound_name)

    if not compound_data:
        return 0.5, "Unknown compound"

    boiling_point = compound_data.get('boiling_point')
    if boiling_point is None:
        # Compounds requiring combustion
        if interface_temp >= 600:
            return 1.0, "Combustion release"
        else:
            return 0.0, f"Requires combustion (>600°C)"

    # Filter Rule: Zero out compounds above interface temp
    if boiling_point > interface_temp:
        return 0.0, f"Below activation ({boiling_point}°C)"

    # Partial release curve for compounds near threshold
    temp_margin = interface_temp - boiling_point
    if temp_margin < 10:
        # Gradual release within 10°C of boiling point
        availability = 0.5 + (temp_margin / 20)
        return availability, f"Partial release ({availability*100:.0f}%)"

    # Full release
    return 1.0, "Full release"

# ========================================================================
# INTEGRATED SCORING WITH DATABASE QUERIES
# ========================================================================

def _score_product_integrated(self, product: Dict, user_profile: Dict,
                              weights: Dict, interface_temp: float) -> Dict:
    """
```

```
Score product with live database queries and all logic blocks
"""
terpene_score = 0.0
cannabinoid_score = 0.0
modifier_score = 0.0
penalties = []
activation_details = []

grow_style = product.get('growStyle', 'hydroponic')

# --- TERPENE SCORING WITH CPI + TRE ---
if product.get('terpenes'):
    for terpene in product['terpenes']:
        # LOGIC BLOCK A: Apply cultivation modifiers
        base_conc = terpene['concentration']
        modified_conc = self._apply_cultivation_modifiers(
            terpene['name'], 'terpene', grow_style, base_conc
        )

        # LOGIC BLOCK B: Check thermal availability
        thermal_avail, thermal_status = self._calculate_thermal_availability(
            terpene['name'], 'terpene', interface_temp
        )

        if thermal_avail == 0:
            activation_details.append(f"{terpene['name']}: {thermal_status}")
            continue

        # Query live efficacy from database
        efficacy = self._get_efficacy_with_severity_integrated(
            terpene['name'], user_profile['conditions'], 'terpene'
        )

        # Apply sensitivity boost
        if terpene['name'] in user_profile.get('sensitiveTerpenes', []):
            efficacy *= 1.5

        # Calculate score with all modifiers
        norm_conc = min(modified_conc / 100, 1.0)
        contribution = efficacy * norm_conc * thermal_avail
        terpene_score += contribution

        if thermal_avail < 1.0:
            activation_details.append(f"{terpene['name']}: {thermal_status}")

# --- CANNABINOID SCORING WITH CPI + TRE + THRESHOLDS ---
if product.get('cannabinoids'):
    for cannabinoid in product['cannabinoids']:
        # LOGIC BLOCK A: Cultivation modifiers
        base_pct = cannabinoid['percentage']
```

```
        modified_pct = self._apply_cultivation_modifiers(
            cannabinoid['name'], 'cannabinoid', grow_style, base_pct
        )

        # LOGIC BLOCK B: Thermal availability
        thermal_avail, thermal_status = self._calculate_thermal_availability(
            cannabinoid['name'], 'cannabinoid', interface_temp
        )

        if thermal_avail == 0:
            activation_details.append(f"{cannabinoid['name']}: {thermal_status}")
            continue

        # Query live efficacy
        efficacy = self._get_efficacy_with_severity_integrated(
            cannabinoid['name'], user_profile['conditions'], 'cannabinoid'
        )

        # Biphasic threshold checks
        thresholds = user_profile.get('thresholds', {})

        if cannabinoid['name'] == "THC":
            max_thc = thresholds.get('maxTHC')
            if max_thc and modified_pct > max_thc:
                efficacy = -abs(efficacy) * 0.5
                penalties.append(
                    f"THC exceeds threshold ({modified_pct:.1f}% > {max_thc}%)"
                )

        if cannabinoid['name'] == "CBD":
            min_cbd = thresholds.get('minCBD')
            if min_cbd and modified_pct < min_cbd:
                efficacy *= 0.7
                penalties.append("CBD below preferred minimum")

        # Calculate contribution
        norm_amount = min(modified_pct / 100, 1.0)
        contribution = efficacy * norm_amount * thermal_avail
        cannabinoid_score += contribution

# --- FLAVONOID SCORING WITH CPI + TRE + POTENCY MULTIPLIERS ---
if product.get('flavonoids'):
    for flavonoid in product['flavonoids']:
        # Get flavonoid data from database
        flav_data = self.pharma_db.get_flavonoid(flavonoid['name'])
        if not flav_data:
            continue

        # LOGIC BLOCK A: Cultivation modifiers
        base_presence = 1.0  # Binary presence
```

```
            modified_presence = self._apply_cultivation_modifiers(
                flavonoid['name'], 'flavonoid', grow_style, base_presence
            )

            # LOGIC BLOCK B: Thermal availability
            thermal_avail, thermal_status = self._calculate_thermal_availability(
                flavonoid['name'], 'flavonoid', interface_temp
            )

            if thermal_avail == 0:
                activation_details.append(f"{flavonoid['name']}: {thermal_status}")
                continue

            # Query live efficacy
            efficacy = self._get_efficacy_with_severity_integrated(
                flavonoid['name'], user_profile['conditions'], 'flavonoid'
            )

            # Apply potency multiplier (e.g., Cannflavin A = 30x aspirin)
            potency_mult = flav_data.get('potency_multiplier', 1.0)

            # Special case: Cannflavin A pain relief
            if flavonoid['name'] == "Cannflavin A" and thermal_avail > 0:
                for condition in user_profile['conditions']:
                    if condition['name'] == "Pain":
                        # Apply 30x multiplier for pain
                        efficacy *= potency_mult
                        activation_details.append(
                            f"Cannflavin A: Pain relief multiplied by {potency_mult}x (30x aspirin potency) at
{interface_temp}°C"
                        )
                        break

            # Calculate contribution
            contribution = efficacy * modified_presence * thermal_avail
            modifier_score += contribution

        # Delivery method preference
        if product.get('deliveryMethod') in user_profile.get('preferredDelivery', []):
            modifier_score += 0.3

        # Grow style preference
        if product.get('growStyle') == user_profile.get('preferredGrowStyle'):
            modifier_score += 0.2

        # --- HISTORY FACTOR ---
        history_factor = self._get_history_factor_integrated(product, user_profile)

        # Calculate raw score
        raw_score = (
```

```
        (weights['terpene'] * terpene_score) +
        (weights['cannabinoid'] * cannabinoid_score) +
        (weights['flavonoid'] * modifier_score)
    ) * (1 + history_factor)

    raw_score = max(0, raw_score)

    return {
        'id': product['id'],
        'name': product['name'],
        'raw': raw_score,
        'components': {
            'terpene': terpene_score,
            'cannabinoid': cannabinoid_score,
            'modifier': modifier_score,
            'historyBoost': history_factor
        },
        'penalties': penalties,
        'activationDetails': activation_details,
        'rationale': None,
        'safetyWarnings': []
    }

# ============================================================================
# DATABASE-INTEGRATED EFFICACY LOOKUP
# ============================================================================

def _get_efficacy_with_severity_integrated(self, compound_name: str,
                                           conditions: List[Dict],
                                           compound_class: str) -> float:
    """
    Query live efficacy from pharmacognosy database with severity weighting
    """
    if not conditions:
        return 0.5

    weighted_efficacy = 0.0
    total_severity = 0.0

    for condition in conditions:
        # Query from pharmacognosy database
        if compound_class == "terpene":
            compound_data = self.pharma_db.get_terpene(compound_name)
        elif compound_class == "cannabinoid":
            compound_data = self.pharma_db.get_cannabinoid(compound_name)
        elif compound_class == "flavonoid":
            compound_data = self.pharma_db.get_flavonoid(compound_name)
        else:
            continue
```

```
        if not compound_data:
            continue

        # Also check knowledge base for condition-specific efficacy
        efficacy = self.kb.lookup_efficacy(
            compound_name, condition['name'], compound_class
        )

        severity = condition['severity']
        weighted_efficacy += efficacy * severity
        total_severity += severity

    if total_severity > 0:
        return min(weighted_efficacy / total_severity, 1.0)
    return 0.5

# ============================================================================
# ADAPTIVE LEARNING WITH FEEDBACK
# ============================================================================

def _get_adaptive_weights(self, user_profile: Dict) -> Dict:
    """
    Evolve weights based on user feedback history
    Learning Rate: 0.1 shifts toward successful chemical classes
    """
    weights = self.BASE_WEIGHTS.copy()

    history = user_profile.get('history', [])
    feedback_count = user_profile.get('feedbackCount', 0)

    if feedback_count < 5 or not history:
        return weights

    # Analyze positive feedback (rating >= 4)
    terpene_success = 0.0
    cannabinoid_success = 0.0
    flavonoid_success = 0.0
    success_count = 0

    for item in history:
        if item.get('rating', 0) >= 4:
            terpene_success += item.get('terpeneScore', 0)
            cannabinoid_success += item.get('cannabinoidScore', 0)
            flavonoid_success += item.get('flavonoidScore', 0)
            success_count += 1

    if success_count > 0:
        avg_terp = terpene_success / success_count
        avg_canna = cannabinoid_success / success_count
        avg_flav = flavonoid_success / success_count
```

```
        total = avg_terp + avg_canna + avg_flav

        if total > 0:
            # Apply learning rate to shift weights
            new_terp = 0.5 + (avg_terp / total - 0.5) * self.LEARNING_RATE
            new_canna = 0.3 + (avg_canna / total - 0.3) * self.LEARNING_RATE
            new_flav = 0.2 + (avg_flav / total - 0.2) * self.LEARNING_RATE

            # Normalize to sum to 1.0
            weight_sum = new_terp + new_canna + new_flav
            weights['terpene'] = new_terp / weight_sum
            weights['cannabinoid'] = new_canna / weight_sum
            weights['flavonoid'] = new_flav / weight_sum

    return weights

def _get_history_factor_integrated(self, product: Dict,
                                   user_profile: Dict) -> float:
    """
    Calculate boost/penalty from user history
    """
    history = user_profile.get('history', [])
    if not history:
```