Comprehensive Prompt: Implementing Nutrition Level and Risk Level Functionality in NutriMap Android

Objective

Implement the **exact same** nutrition level and risk level calculation system from the NutriMap Desktop app into the NutriMap Android app. This includes WHO Growth Standards z-score calculation, nutrition level classification, risk level scoring, and dynamic recalculation from visit data.

Part 1: Data Models

1.1 Update Visit Entity/Model

Ensure your Visit model has these fields (if not already present):

```
// Visit.kt (or VisitEntity if using Room)
data class Visit(
    val visitId: Int = 0,
    val childId: Int,
    val visitDate: String,          // Format: "yyyy-MM-dd"
    val weightKg: Double,           // Weight in kilograms
    val heightCm: Double,           // Height in centimeters
    val muacMm: Int,                // MUAC in MILLIMETERS (e.g., 125 = 12.5 cm)
    val riskLevel: String = "N/A",  // "high", "medium", "low", or "N/A"
    val notes: String? = null,
    val createdAt: String? = null,
    val updatedAt: String? = null,
    val enteredBy: Int? = null,
    val deleted: Boolean = false
)
```

1.2 Ensure Child Model Has Required Fields

```
// Child.kt
data class Child(
    val id: Int = 0,
    val fullName: String,
    val fathersName: String? = null,
    val mothersName: String? = null,
    val contactNumber: String? = null,
    val division: String? = null,
    val district: String? = null,
    val upazilla: String? = null,
    val unionName: String? = null,
    val branchId: String? = null,
    val branchName: String? = null,
    val lastVisit: String? = null,
    val gender: String,             // "Male" or "Female"
    val dateOfBirth: String         // Format: "yyyy-MM-dd"
)
```

Part 2: WHO Growth Standards Utility Class

Create `WhoGrowthStandards.kt` file:

```
package com.example.nutrimap.util

/**
 * WHO Child Growth Standards - Weight-for-Height LMS Reference Data
 *
 * Contains L, M, S parameters for calculating weight-for-height z-scores (WHZ)
 * for children aged 6-59 months using the WHO LMS method.
 *
 * Formula: z = [ (X/M)^L - 1 ] / (L * S)
 * Where: X = child's weight, L, M, S = reference values for given sex and height
 *
 * Data source: WHO Child Growth Standards
 * Height range: 45-120 cm (covering 6-59 months age range)
 */
object WhoGrowthStandards {

    data class LmsParams(val L: Double, val M: Double, val S: Double)

    // WHO Weight-for-Height reference data for BOYS (height in cm -> LMS)
    private val BOYS_WFH: Map<Double, LmsParams> = mapOf(
        // Height 45-60 cm (typical for infants 0-6 months)
        45.0 to LmsParams(-0.3521, 2.441, 0.09182),
        46.0 to LmsParams(-0.3521, 2.528, 0.09153),
        47.0 to LmsParams(-0.3521, 2.618, 0.09124),
        48.0 to LmsParams(-0.3521, 2.711, 0.09094),
        49.0 to LmsParams(-0.3521, 2.807, 0.09065),
        50.0 to LmsParams(-0.3521, 2.906, 0.09036),
        51.0 to LmsParams(-0.3521, 3.010, 0.09007),
        52.0 to LmsParams(-0.3521, 3.117, 0.08977),
        53.0 to LmsParams(-0.3521, 3.227, 0.08948),
        54.0 to LmsParams(-0.3521, 3.341, 0.08919),
        55.0 to LmsParams(-0.3521, 3.459, 0.08889),
        56.0 to LmsParams(-0.3521, 3.581, 0.08860),
        57.0 to LmsParams(-0.3521, 3.708, 0.08831),
        58.0 to LmsParams(-0.3521, 3.840, 0.08802),
        59.0 to LmsParams(-0.3521, 3.976, 0.08773),
        60.0 to LmsParams(-0.3521, 4.117, 0.08744),
        // Height 61-75 cm (typical for 6-12 months)
        61.0 to LmsParams(-0.3521, 4.263, 0.08716),
        62.0 to LmsParams(-0.3521, 4.413, 0.08687),
        63.0 to LmsParams(-0.3521, 4.565, 0.08659),
        64.0 to LmsParams(-0.3521, 4.720, 0.08631),
        65.0 to LmsParams(-0.3521, 4.877, 0.08603),
        66.0 to LmsParams(-0.3521, 5.037, 0.08576),
        67.0 to LmsParams(-0.3521, 5.199, 0.08549),
        68.0 to LmsParams(-0.3521, 5.364, 0.08522),
        69.0 to LmsParams(-0.3521, 5.532, 0.08495),
        70.0 to LmsParams(-0.3521, 5.703, 0.08469),
        71.0 to LmsParams(-0.3521, 5.877, 0.08443),
        72.0 to LmsParams(-0.3521, 6.053, 0.08418),
        73.0 to LmsParams(-0.3521, 6.231, 0.08393),
        74.0 to LmsParams(-0.3521, 6.411, 0.08369),
```

```
    75.0 to LmsParams(-0.3521, 6.593, 0.08345),
    // Height 76-90 cm (typical for 12-24 months)
    76.0 to LmsParams(-0.3521, 6.777, 0.08321),
    77.0 to LmsParams(-0.3521, 6.963, 0.08298),
    78.0 to LmsParams(-0.3521, 7.149, 0.08276),
    79.0 to LmsParams(-0.3521, 7.337, 0.08254),
    80.0 to LmsParams(-0.3521, 7.527, 0.08232),
    81.0 to LmsParams(-0.3521, 7.719, 0.08211),
    82.0 to LmsParams(-0.3521, 7.913, 0.08190),
    83.0 to LmsParams(-0.3521, 8.109, 0.08170),
    84.0 to LmsParams(-0.3521, 8.308, 0.08150),
    85.0 to LmsParams(-0.3521, 8.509, 0.08131),
    86.0 to LmsParams(-0.3521, 8.714, 0.08112),
    87.0 to LmsParams(-0.3521, 8.922, 0.08094),
    88.0 to LmsParams(-0.3521, 9.134, 0.08076),
    89.0 to LmsParams(-0.3521, 9.350, 0.08059),
    90.0 to LmsParams(-0.3521, 9.570, 0.08042),
    // Height 91-105 cm (typical for 24-48 months)
    91.0 to LmsParams(-0.3521, 9.795, 0.08025),
    92.0 to LmsParams(-0.3521, 10.024, 0.08009),
    93.0 to LmsParams(-0.3521, 10.258, 0.07993),
    94.0 to LmsParams(-0.3521, 10.496, 0.07978),
    95.0 to LmsParams(-0.3521, 10.739, 0.07963),
    96.0 to LmsParams(-0.3521, 10.987, 0.07948),
    97.0 to LmsParams(-0.3521, 11.240, 0.07934),
    98.0 to LmsParams(-0.3521, 11.498, 0.07920),
    99.0 to LmsParams(-0.3521, 11.761, 0.07907),
    100.0 to LmsParams(-0.3521, 12.029, 0.07894),
    101.0 to LmsParams(-0.3521, 12.302, 0.07881),
    102.0 to LmsParams(-0.3521, 12.580, 0.07869),
    103.0 to LmsParams(-0.3521, 12.864, 0.07857),
    104.0 to LmsParams(-0.3521, 13.153, 0.07845),
    105.0 to LmsParams(-0.3521, 13.448, 0.07834),
    // Height 106-120 cm (typical for 48-59 months)
    106.0 to LmsParams(-0.3521, 13.749, 0.07823),
    107.0 to LmsParams(-0.3521, 14.056, 0.07813),
    108.0 to LmsParams(-0.3521, 14.369, 0.07803),
    109.0 to LmsParams(-0.3521, 14.688, 0.07793),
    110.0 to LmsParams(-0.3521, 15.014, 0.07783),
    111.0 to LmsParams(-0.3521, 15.346, 0.07774),
    112.0 to LmsParams(-0.3521, 15.685, 0.07765),
    113.0 to LmsParams(-0.3521, 16.030, 0.07757),
    114.0 to LmsParams(-0.3521, 16.382, 0.07749),
    115.0 to LmsParams(-0.3521, 16.741, 0.07741),
    116.0 to LmsParams(-0.3521, 17.107, 0.07733),
    117.0 to LmsParams(-0.3521, 17.480, 0.07726),
    118.0 to LmsParams(-0.3521, 17.860, 0.07719),
    119.0 to LmsParams(-0.3521, 18.247, 0.07713),
    120.0 to LmsParams(-0.3521, 18.641, 0.07707)
)

// WHO Weight-for-Height reference data for GIRLS
private val GIRLS_WFH: Map<Double, LmsParams> = mapOf(
    // Height 45-60 cm
    45.0 to LmsParams(-0.3833, 2.343, 0.09029),
    46.0 to LmsParams(-0.3833, 2.421, 0.09003),
```

```
    47.0 to LmsParams(-0.3833, 2.503, 0.08977),
    48.0 to LmsParams(-0.3833, 2.588, 0.08951),
    49.0 to LmsParams(-0.3833, 2.676, 0.08925),
    50.0 to LmsParams(-0.3833, 2.768, 0.08899),
    51.0 to LmsParams(-0.3833, 2.863, 0.08873),
    52.0 to LmsParams(-0.3833, 2.962, 0.08847),
    53.0 to LmsParams(-0.3833, 3.064, 0.08821),
    54.0 to LmsParams(-0.3833, 3.170, 0.08795),
    55.0 to LmsParams(-0.3833, 3.281, 0.08769),
    56.0 to LmsParams(-0.3833, 3.396, 0.08743),
    57.0 to LmsParams(-0.3833, 3.515, 0.08717),
    58.0 to LmsParams(-0.3833, 3.638, 0.08691),
    59.0 to LmsParams(-0.3833, 3.766, 0.08665),
    60.0 to LmsParams(-0.3833, 3.899, 0.08639),
    // Height 61–75 cm
    61.0 to LmsParams(-0.3833, 4.036, 0.08614),
    62.0 to LmsParams(-0.3833, 4.177, 0.08589),
    63.0 to LmsParams(-0.3833, 4.321, 0.08564),
    64.0 to LmsParams(-0.3833, 4.469, 0.08539),
    65.0 to LmsParams(-0.3833, 4.620, 0.08515),
    66.0 to LmsParams(-0.3833, 4.773, 0.08491),
    67.0 to LmsParams(-0.3833, 4.929, 0.08468),
    68.0 to LmsParams(-0.3833, 5.088, 0.08445),
    69.0 to LmsParams(-0.3833, 5.251, 0.08422),
    70.0 to LmsParams(-0.3833, 5.418, 0.08400),
    71.0 to LmsParams(-0.3833, 5.588, 0.08379),
    72.0 to LmsParams(-0.3833, 5.762, 0.08358),
    73.0 to LmsParams(-0.3833, 5.939, 0.08338),
    74.0 to LmsParams(-0.3833, 6.120, 0.08318),
    75.0 to LmsParams(-0.3833, 6.303, 0.08299),
    // Height 76–90 cm
    76.0 to LmsParams(-0.3833, 6.490, 0.08280),
    77.0 to LmsParams(-0.3833, 6.679, 0.08262),
    78.0 to LmsParams(-0.3833, 6.871, 0.08245),
    79.0 to LmsParams(-0.3833, 7.066, 0.08228),
    80.0 to LmsParams(-0.3833, 7.264, 0.08211),
    81.0 to LmsParams(-0.3833, 7.464, 0.08195),
    82.0 to LmsParams(-0.3833, 7.667, 0.08180),
    83.0 to LmsParams(-0.3833, 7.873, 0.08165),
    84.0 to LmsParams(-0.3833, 8.082, 0.08151),
    85.0 to LmsParams(-0.3833, 8.293, 0.08137),
    86.0 to LmsParams(-0.3833, 8.508, 0.08124),
    87.0 to LmsParams(-0.3833, 8.725, 0.08111),
    88.0 to LmsParams(-0.3833, 8.946, 0.08099),
    89.0 to LmsParams(-0.3833, 9.170, 0.08088),
    90.0 to LmsParams(-0.3833, 9.397, 0.08076),
    // Height 91–105 cm
    91.0 to LmsParams(-0.3833, 9.628, 0.08066),
    92.0 to LmsParams(-0.3833, 9.862, 0.08055),
    93.0 to LmsParams(-0.3833, 10.099, 0.08046),
    94.0 to LmsParams(-0.3833, 10.340, 0.08036),
    95.0 to LmsParams(-0.3833, 10.584, 0.08027),
    96.0 to LmsParams(-0.3833, 10.832, 0.08019),
    97.0 to LmsParams(-0.3833, 11.083, 0.08011),
    98.0 to LmsParams(-0.3833, 11.338, 0.08003),
    99.0 to LmsParams(-0.3833, 11.597, 0.07996),
```

```
    100.0 to LmsParams(-0.3833, 11.859, 0.07989),
    101.0 to LmsParams(-0.3833, 12.125, 0.07983),
    102.0 to LmsParams(-0.3833, 12.394, 0.07977),
    103.0 to LmsParams(-0.3833, 12.668, 0.07971),
    104.0 to LmsParams(-0.3833, 12.946, 0.07966),
    105.0 to LmsParams(-0.3833, 13.228, 0.07961),
    // Height 106-120 cm
    106.0 to LmsParams(-0.3833, 13.515, 0.07957),
    107.0 to LmsParams(-0.3833, 13.806, 0.07953),
    108.0 to LmsParams(-0.3833, 14.102, 0.07949),
    109.0 to LmsParams(-0.3833, 14.402, 0.07946),
    110.0 to LmsParams(-0.3833, 14.707, 0.07943),
    111.0 to LmsParams(-0.3833, 15.018, 0.07941),
    112.0 to LmsParams(-0.3833, 15.334, 0.07939),
    113.0 to LmsParams(-0.3833, 15.656, 0.07937),
    114.0 to LmsParams(-0.3833, 15.983, 0.07936),
    115.0 to LmsParams(-0.3833, 16.316, 0.07935),
    116.0 to LmsParams(-0.3833, 16.655, 0.07934),
    117.0 to LmsParams(-0.3833, 17.000, 0.07934),
    118.0 to LmsParams(-0.3833, 17.352, 0.07934),
    119.0 to LmsParams(-0.3833, 17.710, 0.07935),
    120.0 to LmsParams(-0.3833, 18.075, 0.07936)
)

/**
 * Get LMS parameters for weight-for-height based on sex and height.
 * Uses nearest-neighbor interpolation if exact height not found.
 */
fun getWeightForHeightLms(sex: String, heightCm: Double): LmsParams? {
    val refData = if (sex.equals("M", ignoreCase = true)) BOYS_WFH else GIRLS_WFH

    // Round to nearest whole cm for lookup
    val roundedHeight = kotlin.math.round(heightCm)

    // Clamp to valid range
    if (roundedHeight < 45.0 || roundedHeight > 120.0) {
        return null
    }

    // Direct lookup
    refData[roundedHeight]?.let { return it }

    // Try interpolation between two nearest heights
    val lowerHeight = kotlin.math.floor(heightCm)
    val upperHeight = kotlin.math.ceil(heightCm)

    val lowerParams = refData[lowerHeight]
    val upperParams = refData[upperHeight]

    return when {
        lowerParams != null && upperParams != null -> {
            val t = heightCm - lowerHeight
            LmsParams(
                L = lowerParams.L + t * (upperParams.L - lowerParams.L),
                M = lowerParams.M + t * (upperParams.M - lowerParams.M),
                S = lowerParams.S + t * (upperParams.S - lowerParams.S)
```

```
            )
        }
        lowerParams != null -> lowerParams
        else -> upperParams
    }
}

/**
 * Calculate z-score using the LMS method.
 * Formula: z = [ (X/M)^L - 1 ] / (L * S)
 */
fun calculateZScore(measurement: Double, params: LmsParams?): Double {
    if (params == null || measurement <= 0) {
        return Double.NaN
    }

    val (L, M, S) = params

    return if (L == 0.0) {
        // Special case when L = 0: z = ln(X/M) / S
        kotlin.math.ln(measurement / M) / S
    } else {
        // Standard LMS formula: z = [ (X/M)^L - 1 ] / (L * S)
        (kotlin.math.pow(measurement / M, L) - 1) / (L * S)
    }
}

/**
 * Compute Weight-for-Height Z-score (WHZ) using WHO standards.
 */
fun computeWhzZScore(ageMonths: Int, sex: String, heightCm: Double, weightKg: Double): Double {
    // Validate inputs
    if (!sex.equals("M", ignoreCase = true) && !sex.equals("F", ignoreCase = true)) {
        return Double.NaN
    }

    if (heightCm <= 0 || weightKg <= 0) {
        return Double.NaN
    }

    // Height out of reference range
    if (heightCm < 45.0 || heightCm > 120.0) {
        return Double.NaN
    }

    // Get LMS parameters for the given sex and height
    val params = getWeightForHeightLms(sex, heightCm) ?: return Double.NaN

    // Calculate z-score
    return calculateZScore(weightKg, params)
}

/**
 * Convert gender string to sex code.
 * "Male", "Female", "M", "F" -> "M" or "F"
 */
```

```
    fun genderToSex(gender: String?): String? {
        if (gender == null) return null
        val g = gender.trim().uppercase()
        return when {
            g.startsWith("M") -> "M"
            g.startsWith("F") -> "F"
            else -> null
        }
    }
}
```

Part 3: Nutrition Risk Calculator Utility Class

Create `NutritionRiskCalculator.kt`:

```
package com.example.nutrimap.util

import java.time.LocalDate
import java.time.Period
import java.time.format.DateTimeFormatter

/**
 * Utility class for calculating nutrition levels and risk levels.
 *
 * Nutrition Level (A1 + A2 Combined):
 * - Severe malnutrition: MUAC < 11.5 cm OR z-score < -3
 * - Moderate malnutrition: MUAC 11.5-12.5 cm OR z-score -3 to -2
 * - Normal: Otherwise
 *
 * Risk Level (B2 Scoring System):
 * - Based on nutrition level, age, borderline MUAC, and trend factors
 * - Points >= 4: High, Points 2-3: Medium, Points <= 1: Low
 */
object NutritionRiskCalculator {

    // MUAC thresholds in centimeters
    const val MUAC_SEVERE_THRESHOLD_CM = 11.5
    const val MUAC_MODERATE_THRESHOLD_CM = 12.5

    // Z-score thresholds
    const val ZSCORE_SEVERE_THRESHOLD = -3.0
    const val ZSCORE_MODERATE_THRESHOLD = -2.0

    // Nutrition level strings (exact format as specified)
    const val NUTRITION_SEVERE = "severe malnutrition"
    const val NUTRITION_MODERATE = "moderate malnutrition"
    const val NUTRITION_NORMAL = "normal"

    // Risk level strings
    const val RISK_HIGH = "high"
    const val RISK_MEDIUM = "medium"
    const val RISK_LOW = "low"
    const val RISK_NA = "N/A"

    /**
```

```
 * Result class containing nutrition level, risk level, and computed z-score.
 */
data class NutritionRiskResult(
    val nutritionLevel: String,
    val riskLevel: String,
    val zScore: Double
) {
    val hasValidZScore: Boolean get() = !zScore.isNaN()

    val nutritionLevelDisplay: String get() = getNutritionLevelDisplay(nutritionLevel)
    val riskLevelDisplay: String get() = getRiskLevelDisplay(riskLevel)
    val zScoreDisplay: String get() = if (zScore.isNaN()) "N/A" else String.format("%.2f", zScore)
}

/**
 * Classify nutrition level based on MUAC (cm) and WHZ/BAZ z-score.
 * Uses the LATEST visit data only.
 *
 * Rules:
 * - Severe malnutrition if: muacCm < 11.5 OR zScore < -3
 * - Moderate malnutrition if: (11.5 <= muacCm < 12.5) OR (-3 <= zScore < -2)
 * - Normal otherwise
 */
fun classifyNutritionLevel(muacCm: Double, zScore: Double? = null): String {
    // Check for invalid MUAC
    if (muacCm <= 0) {
        // If MUAC is invalid, rely solely on z-score if available
        if (zScore != null) {
            return when {
                zScore < ZSCORE_SEVERE_THRESHOLD -> NUTRITION_SEVERE
                zScore < ZSCORE_MODERATE_THRESHOLD -> NUTRITION_MODERATE
                else -> NUTRITION_NORMAL
            }
        }
        return NUTRITION_NORMAL // Default if no valid data
    }

    // Check severe malnutrition conditions
    if (muacCm < MUAC_SEVERE_THRESHOLD_CM) {
        return NUTRITION_SEVERE
    }
    if (zScore != null && zScore < ZSCORE_SEVERE_THRESHOLD) {
        return NUTRITION_SEVERE
    }

    // Check moderate malnutrition conditions
    if (muacCm < MUAC_MODERATE_THRESHOLD_CM) {
        return NUTRITION_MODERATE
    }
    if (zScore != null && zScore < ZSCORE_MODERATE_THRESHOLD) {
        return NUTRITION_MODERATE
    }

    // Normal
    return NUTRITION_NORMAL
}
```

```kotlin
/**
 * Classify risk level using the B2 scoring system.
 * Uses the LATEST visit data for current values, and previous visit for trend analysis.
 *
 * Risk points logic:
 * - Base on nutritionLevel: severe=+3, moderate=+2, normal=+1
 * - Age factor: if ageMonths < 24 -> +1
 * - Borderline MUAC: 11.5–11.9 cm -> +1, 12.5–12.9 cm -> +1
 * - Trend: MUAC drop >= 0.5 cm -> +1
 * - Trend: Weight loss >= 5% -> +1
 *
 * Final mapping: points >= 4 -> high, 2–3 -> medium, <= 1 -> low
 */
fun classifyRiskLevel(
    nutritionLevel: String,
    ageMonths: Int,
    muacCm: Double,
    muacPrevCm: Double?,
    weightKg: Double,
    weightPrevKg: Double?
): String {
    var riskPoints = 0

    // Base points from nutrition level
    riskPoints += when (nutritionLevel) {
        NUTRITION_SEVERE -> 3
        NUTRITION_MODERATE -> 2
        else -> 1
    }

    // Age factor: children under 24 months are at higher risk
    if (ageMonths < 24) {
        riskPoints += 1
    }

    // Borderline MUAC checks
    // Borderline severe: 11.5 <= muacCm < 11.9
    if (muacCm >= 11.5 && muacCm < 11.9) {
        riskPoints += 1
    }
    // Borderline moderate: 12.5 <= muacCm < 12.9
    if (muacCm >= 12.5 && muacCm < 12.9) {
        riskPoints += 1
    }

    // Trend factor: MUAC decline >= 0.5 cm (only if previous data exists)
    if (muacPrevCm != null && muacPrevCm > 0) {
        val muacDrop = muacPrevCm - muacCm
        if (muacDrop >= 0.5) {
            riskPoints += 1
        }
    }

    // Trend factor: Weight loss >= 5% (only if previous data exists)
    if (weightPrevKg != null && weightPrevKg > 0) {
```

```
        val weightLossPercent = (weightPrevKg - weightKg) / weightPrevKg
        if (weightLossPercent >= 0.05) {
            riskPoints += 1
        }
    }

    // Final risk level mapping
    return when {
        riskPoints >= 4 -> RISK_HIGH
        riskPoints >= 2 -> RISK_MEDIUM
        else -> RISK_LOW
    }
}

/**
 * Calculate age in months from date of birth string (calculates age as of today).
 */
fun calculateAgeInMonths(dateOfBirth: String?): Int {
    if (dateOfBirth.isNullOrEmpty()) return -1
    return try {
        val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")
        val dob = LocalDate.parse(dateOfBirth, formatter)
        val now = LocalDate.now()
        val period = Period.between(dob, now)
        period.years * 12 + period.months
    } catch (e: Exception) {
        -1
    }
}

/**
 * Calculate age in months at a specific date (visit date).
 * This is the preferred method for calculating age at time of visit.
 */
fun calculateAgeInMonths(birthDate: LocalDate, visitDate: LocalDate): Int {
    if (visitDate.isBefore(birthDate)) return -1
    val period = Period.between(birthDate, visitDate)
    return period.years * 12 + period.months
}

/**
 * Calculate age in months from date strings (birth date and visit date).
 */
fun calculateAgeInMonths(birthDateStr: String?, visitDateStr: String?): Int {
    if (birthDateStr.isNullOrEmpty() || visitDateStr.isNullOrEmpty()) return -1
    return try {
        val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")
        val birthDate = LocalDate.parse(birthDateStr, formatter)
        val visitDate = LocalDate.parse(visitDateStr, formatter)
        calculateAgeInMonths(birthDate, visitDate)
    } catch (e: Exception) {
        -1
    }
}

/**
```

```kotlin
 * Convert MUAC from millimeters to centimeters.
 */
fun muacMmToCm(muacMm: Int): Double = muacMm / 10.0
fun muacMmToCm(muacMm: Double): Double = muacMm / 10.0

/**
 * Get display-friendly nutrition level (capitalized).
 */
fun getNutritionLevelDisplay(nutritionLevel: String?): String {
    return when (nutritionLevel) {
        NUTRITION_SEVERE -> "Severe Malnutrition"
        NUTRITION_MODERATE -> "Moderate Malnutrition"
        NUTRITION_NORMAL -> "Normal"
        else -> "N/A"
    }
}

/**
 * Get display-friendly risk level (uppercase).
 */
fun getRiskLevelDisplay(riskLevel: String?): String {
    return riskLevel?.uppercase() ?: "N/A"
}

/**
 * Compute Weight-for-Height Z-score (WHZ) using WHO Growth Standards.
 * This is a convenience wrapper around WhoGrowthStandards.computeWhzZScore().
 */
fun computeWhzZScore(ageMonths: Int, sex: String, heightCm: Double, weightKg: Double): Double {
    // Convert gender string to sex code
    val sexCode = WhoGrowthStandards.genderToSex(sex) ?: return Double.NaN
    return WhoGrowthStandards.computeWhzZScore(ageMonths, sexCode, heightCm, weightKg)
}

/**
 * Classify nutrition level from raw measurement data.
 * Automatically computes z-score using WHO Growth Standards.
 * Uses LATEST visit data only.
 */
fun classifyNutritionLevelFromRawData(
    ageMonths: Int,
    sex: String,
    heightCm: Double,
    weightKg: Double,
    muacCm: Double
): String {
    // Compute z-score using WHO Growth Standards
    val zScore = computeWhzZScore(ageMonths, sex, heightCm, weightKg)

    // If z-score is valid, use it with MUAC for classification
    if (!zScore.isNaN()) {
        return classifyNutritionLevel(muacCm, zScore)
    }

    // Fallback to MUAC-only classification if z-score couldn't be computed
    if (muacCm > 0) {
```

```
        return when {
            muacCm < MUAC_SEVERE_THRESHOLD_CM -> NUTRITION_SEVERE
            muacCm < MUAC_MODERATE_THRESHOLD_CM -> NUTRITION_MODERATE
            else -> NUTRITION_NORMAL
        }
    }

    // No valid data - default to normal
    return NUTRITION_NORMAL
}

/**
 * Complete evaluation from visit data.
 * Calculates nutrition level and risk level from Child + Visit entities.
 *
 * THIS IS THE MAIN METHOD TO USE FOR CALCULATING BOTH LEVELS.
 *
 * @param birthDateStr Child's date of birth in "yyyy-MM-dd" format
 * @param visitDateStr Visit date in "yyyy-MM-dd" format
 * @param sex "M" or "F" (or "Male"/"Female")
 * @param heightCm Height in centimeters
 * @param weightKg Weight in kilograms
 * @param muacMm Current MUAC in MILLIMETERS
 * @param muacPrevMm Previous MUAC in MILLIMETERS (nullable)
 * @param weightPrevKg Previous weight in kilograms (nullable)
 * @return NutritionRiskResult containing nutritionLevel, riskLevel, and zScore
 */
fun evaluateFromVisitData(
    birthDateStr: String?,
    visitDateStr: String?,
    sex: String?,
    heightCm: Double,
    weightKg: Double,
    muacMm: Int,
    muacPrevMm: Double?,
    weightPrevKg: Double?
): NutritionRiskResult {
    // Calculate age at time of visit
    var ageMonths = calculateAgeInMonths(birthDateStr, visitDateStr)
    if (ageMonths < 0) {
        ageMonths = 36 // Default if dates invalid
    }

    // Convert MUAC from mm to cm
    val muacCm = muacMmToCm(muacMm)
    val muacPrevCm = muacPrevMm?.let { muacMmToCm(it) }

    // Classify nutrition level
    val nutritionLevel = classifyNutritionLevelFromRawData(
        ageMonths, sex ?: "M", heightCm, weightKg, muacCm
    )

    // Classify risk level
    val riskLevel = classifyRiskLevel(
        nutritionLevel,
        ageMonths,
```

```
            if (muacCm > 0) muacCm else 13.0,
            muacPrevCm,
            weightKg,
            weightPrevKg
        )

        // Compute z-score for informational purposes
        val zScore = computeWhzZScore(ageMonths, sex ?: "M", heightCm, weightKg)

        return NutritionRiskResult(nutritionLevel, riskLevel, zScore)
    }
}
```

Part 4: Update Your ViewModels/Repositories

4.1 When Creating a New Visit

In your `CreateVisitViewModel` or wherever you create visits:

```
fun createVisit(
    child: Child,
    visitDate: String,
    weightKg: Double,
    heightCm: Double,
    muacMm: Int,
    notes: String?
) {
    viewModelScope.launch {
        // Get previous visit for trend analysis
        val previousVisits = visitRepository.getVisitsByChildId(child.id)
        val previousVisit = previousVisits.firstOrNull()

        // Get previous values for trend analysis
        val muacPrevMm = previousVisit?.muacMm?.toDouble()
        val weightPrevKg = previousVisit?.weightKg

        // Calculate nutrition and risk levels
        val result = NutritionRiskCalculator.evaluateFromVisitData(
            birthDateStr = child.dateOfBirth,
            visitDateStr = visitDate,
            sex = child.gender,
            heightCm = heightCm,
            weightKg = weightKg,
            muacMm = muacMm,
            muacPrevMm = muacPrevMm,
            weightPrevKg = weightPrevKg
        )

        // Create visit with calculated risk level
        val newVisit = Visit(
            childId = child.id,
            visitDate = visitDate,
            weightKg = weightKg,
            heightCm = heightCm,
            muacMm = muacMm,
```

```
            riskLevel = result.riskLevel,  // STORE THE CALCULATED RISK LEVEL
            notes = notes
        )

        visitRepository.insertVisit(newVisit)
    }
}
```

4.2 When Editing a Visit

```
fun updateVisit(
    child: Child,
    existingVisit: Visit,
    visitDate: String,
    weightKg: Double,
    heightCm: Double,
    muacMm: Int,
    notes: String?
) {
    viewModelScope.launch {
        // Get all visits except the one being edited
        val allVisits = visitRepository.getVisitsByChildId(child.id)
        val previousVisit = allVisits.filter { it.visitId != existingVisit.visitId }.firstOrNull()

        // Get previous values for trend analysis
        val muacPrevMm = previousVisit?.muacMm?.toDouble()
        val weightPrevKg = previousVisit?.weightKg

        // Recalculate nutrition and risk levels
        val result = NutritionRiskCalculator.evaluateFromVisitData(
            birthDateStr = child.dateOfBirth,
            visitDateStr = visitDate,
            sex = child.gender,
            heightCm = heightCm,
            weightKg = weightKg,
            muacMm = muacMm,
            muacPrevMm = muacPrevMm,
            weightPrevKg = weightPrevKg
        )

        // Update visit with recalculated risk level
        val updatedVisit = existingVisit.copy(
            visitDate = visitDate,
            weightKg = weightKg,
            heightCm = heightCm,
            muacMm = muacMm,
            riskLevel = result.riskLevel,
            notes = notes
        )

        visitRepository.updateVisit(updatedVisit)
    }
}
```

Part 5: Display in Child Profile Screen

5.1 ViewModel Logic for Child Profile

```
class ChildProfileViewModel(
    private val childRepository: ChildRepository,
    private val visitRepository: VisitRepository
) : ViewModel() {

    private val _nutritionRiskResult = MutableLiveData<NutritionRiskCalculator.NutritionRiskResult?>()
    val nutritionRiskResult: LiveData<NutritionRiskCalculator.NutritionRiskResult?> =
_nutritionRiskResult

    fun loadChildProfile(childId: Int) {
        viewModelScope.launch {
            val child = childRepository.getChildById(childId)
            val visits = visitRepository.getVisitsByChildId(childId)
                .sortedByDescending { it.visitDate }  // Latest first

            if (child != null && visits.isNotEmpty()) {
                val latestVisit = visits[0]
                val previousVisit = visits.getOrNull(1)

                // Get previous values for trend analysis
                val muacPrevMm = previousVisit?.let { if (it.muacMm > 0) it.muacMm.toDouble() else
null }
                val weightPrevKg = previousVisit?.let { if (it.weightKg > 0) it.weightKg else null }

                // RECALCULATE nutrition and risk from latest visit data
                val result = NutritionRiskCalculator.evaluateFromVisitData(
                    birthDateStr = child.dateOfBirth,
                    visitDateStr = latestVisit.visitDate,
                    sex = child.gender,
                    heightCm = latestVisit.heightCm,
                    weightKg = latestVisit.weightKg,
                    muacMm = latestVisit.muacMm,
                    muacPrevMm = muacPrevMm,
                    weightPrevKg = weightPrevKg
                )

                _nutritionRiskResult.value = result
            } else {
                _nutritionRiskResult.value = null
            }
        }
    }
}
```

5.2 UI Display (Jetpack Compose Example)

```
@Composable
fun NutritionRiskBadges(result: NutritionRiskCalculator.NutritionRiskResult?) {
    Row(
        modifier = Modifier.fillMaxWidth(),
```

```
        horizontalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        // Nutrition Level Badge
        StatusBadge(
            label = "Nutrition Level",
            value = result?.nutritionLevelDisplay ?: "N/A",
            color = getNutritionColor(result?.nutritionLevel)
        )

        // Risk Level Badge
        StatusBadge(
            label = "Risk Level",
            value = result?.riskLevelDisplay ?: "N/A",
            color = getRiskColor(result?.riskLevel)
        )
    }
}

@Composable
fun StatusBadge(label: String, value: String, color: Color) {
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        Text(
            text = label,
            style = MaterialTheme.typography.caption,
            color = Color.Gray
        )
        Spacer(modifier = Modifier.height(4.dp))
        Box(
            modifier = Modifier
                .background(color, RoundedCornerShape(12.dp))
                .padding(horizontal = 12.dp, vertical = 4.dp)
        ) {
            Text(
                text = value,
                color = Color.White,
                fontWeight = FontWeight.Bold,
                fontSize = 12.sp
            )
        }
    }
}

// Color helper functions
fun getNutritionColor(nutritionLevel: String?): Color {
    return when (nutritionLevel) {
        NutritionRiskCalculator.NUTRITION_SEVERE -> Color(0xFFE74C3C)  // Red
        NutritionRiskCalculator.NUTRITION_MODERATE -> Color(0xFFF39C12)  // Orange
        NutritionRiskCalculator.NUTRITION_NORMAL -> Color(0xFF27AE60)  // Green
        else -> Color(0xFF95A5A6)  // Gray
    }
}

fun getRiskColor(riskLevel: String?): Color {
    return when (riskLevel?.lowercase()) {
        "high" -> Color(0xFFE74C3C)  // Red
        "medium" -> Color(0xFFF39C12)  // Orange
```

```
        "low" -> Color(0xFF27AE60)  // Green
        else -> Color(0xFF95A5A6)  // Gray
    }
}
```

Part 6: Dashboard Statistics

6.1 Calculate Risk Distribution for Dashboard

```
class DashboardViewModel(
    private val childRepository: ChildRepository,
    private val visitRepository: VisitRepository
) : ViewModel() {

    data class RiskSummary(
        val highRisk: Int = 0,
        val mediumRisk: Int = 0,
        val lowRisk: Int = 0
    )

    private val _riskSummary = MutableLiveData<RiskSummary>()
    val riskSummary: LiveData<RiskSummary> = _riskSummary

    fun loadDashboardStats() {
        viewModelScope.launch {
            val children = childRepository.getAllChildren()
            val allVisits = visitRepository.getAllVisits()

            // Get RECALCULATED risk level for each child based on their latest visit
            val childRisks = getLatestVisitRiskPerChild(children, allVisits)

            // Count by risk level
            var high = 0
            var medium = 0
            var low = 0

            childRisks.values.forEach { risk ->
                when (risk?.lowercase()) {
                    "high" -> high++
                    "medium" -> medium++
                    "low" -> low++
                }
            }

            _riskSummary.value = RiskSummary(high, medium, low)
        }
    }

    private suspend fun getLatestVisitRiskPerChild(
        children: List<Child>,
        visits: List<Visit>
    ): Map<Int, String> {
        val childMap = children.associateBy { it.id }
```

```
        // Group visits by child and find latest + previous
        val visitsByChild = visits.groupBy { it.childId }

        val result = mutableMapOf<Int, String>()

        for ((childId, childVisits) in visitsByChild) {
            val child = childMap[childId] ?: continue

            // Sort visits by date descending (latest first)
            val sortedVisits = childVisits.sortedByDescending { it.visitDate }

            val latestVisit = sortedVisits.firstOrNull() ?: continue
            val previousVisit = sortedVisits.getOrNull(1)

            // Get previous values for trend analysis
            val muacPrevMm = previousVisit?.let {
                if (it.muacMm > 0) it.muacMm.toDouble() else null
            }
            val weightPrevKg = previousVisit?.let {
                if (it.weightKg > 0) it.weightKg else null
            }

            // RECALCULATE risk level (do NOT use stored value)
            val evalResult = NutritionRiskCalculator.evaluateFromVisitData(
                birthDateStr = child.dateOfBirth,
                visitDateStr = latestVisit.visitDate,
                sex = child.gender,
                heightCm = latestVisit.heightCm,
                weightKg = latestVisit.weightKg,
                muacMm = latestVisit.muacMm,
                muacPrevMm = muacPrevMm,
                weightPrevKg = weightPrevKg
            )

            result[childId] = evalResult.riskLevel
        }

        return result
    }
}
```

Part 7: XML Layout Styling (Alternative to Compose)

If using XML layouts, create styles for status badges:

```
<!-- res/values/colors.xml -->
<resources>
    <color name="status_low">#27AE60</color>
    <color name="status_normal">#27AE60</color>
    <color name="status_medium">#F39C12</color>
    <color name="status_moderate">#F39C12</color>
    <color name="status_high">#E74C3C</color>
    <color name="status_severe">#E74C3C</color>
```

```
    <color name="status_na">#95A5A6</color>
</resources>
```

```
<!-- res/drawable/status_badge_high.xml -->
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="@color/status_high" />
    <corners android:radius="12dp" />
    <padding
        android:left="12dp"
        android:right="12dp"
        android:top="4dp"
        android:bottom="4dp" />
</shape>

<!-- Create similar drawables for medium, low, normal, moderate, severe, na -->
```

Part 8: Key Points Summary

✅ Critical Implementation Rules

1. **MUAC Storage**: Always store MUAC in **MILLIMETERS** (e.g., 125 = 12.5 cm)
2. **MUAC Conversion**: Always convert MUAC from mm to cm before calculations: `muacMm / 10.0`
3. **Age Calculation**: Calculate age at **time of visit**, not current age:

   ```
   calculateAgeInMonths(child.dateOfBirth, visit.visitDate)
   ```

4. **Trend Analysis**: Always get previous visit data for trend factors (MUAC drop, weight loss)
5. **Dynamic Recalculation**: For dashboard and profile view, **always recalculate** nutrition and risk levels from the latest visit data - don't rely solely on stored values
6. **Nutrition Level Classification**:
   - Severe: MUAC < 11.5 cm **OR** z-score < -3
   - Moderate: MUAC 11.5-12.5 cm **OR** z-score -3 to -2
   - Normal: Otherwise
7. **Risk Level Scoring** (B2 System):
   - Base points: Severe +3, Moderate +2, Normal +1
   - Age < 24 months: +1
   - Borderline MUAC (11.5-11.9 or 12.5-12.9 cm): +1
   - MUAC drop ≥ 0.5 cm: +1
   - Weight loss ≥ 5%: +1
   - Final: ≥4 = High, 2-3 = Medium, ≤1 = Low
8. **Display Strings**:
   - Nutrition: "Severe Malnutrition", "Moderate Malnutrition", "Normal"
   - Risk: "HIGH", "MEDIUM", "LOW"
9. **Colors** (consistent across app):
   - High/Severe: `#E74C3C` (Red)
   - Medium/Moderate: `#F39C12` (Orange)

- Low/Normal: `#27AE60` (Green)
- N/A: `#95A5A6` (Gray)

Part 9: Testing Checklist

After implementation, verify:

- [ ] Creating a visit calculates and stores correct risk level
- [ ] Editing a visit recalculates risk level
- [ ] Child profile shows dynamically calculated nutrition and risk levels
- [ ] Dashboard counts are based on recalculated (not stored) risk levels
- [ ] Colors match across all screens
- [ ] Z-score calculation works for heights 45-120 cm
- [ ] MUAC-only classification works when z-score unavailable
- [ ] Age is calculated at visit date, not current date
- [ ] Trend factors (MUAC drop, weight loss) correctly influence risk score

---

This prompt provides everything needed to implement the exact same nutrition and risk level calculation system from the desktop app into your Android app. Follow the code examples carefully and ensure all constants and thresholds match exactly.