```
struct AppUser: Codable {
    let id: String
    var trainingCalendar: TrainingCalendar?
    var trainingData: TrainingData? = TrainingData(oneRepMax: [:], workoutHistory: [],
oneRepMaxCount: [:])
    var trainingGoal: TrainingGoal?
    var name: String?
    var age: Int?
    var gender: String?
    var weight: Double?
    var height: Double?
    var workoutFrequency: Int?
    var email: String?
    var isNewUser: Bool?
    var trainingPlanVersion: String?

    enum CodingKeys: String, CodingKey {
        case trainingCalendar,
trainingData ,trainingGoal ,upcomingWorkouts ,id ,name ,age ,gender ,weight ,height ,workoutF
requency ,email ,isNewUser, trainingPlanVersion
    }

    init(id: String, trainingCalendar: TrainingCalendar? = nil, trainingData: TrainingData? =
TrainingData(), trainingGoal: TrainingGoal? = nil, name: String? = nil, age: Int? = nil, gender:
String? = nil, weight: Double? = nil, height: Double? = nil, workoutFrequency: Int? = nil, email:
String?, isNewUser: Bool?, trainingPlanVersion: String?) {
        self.id = id
        self.trainingCalendar = trainingCalendar
        self.trainingData = trainingData
        self.trainingGoal = trainingGoal
        self.name = name
        self.age = age
        self.gender = gender
        self.weight = weight
        self.height = height
        self.workoutFrequency = workoutFrequency
        self.email = email
        self.isNewUser = isNewUser
        self.trainingPlanVersion = trainingPlanVersion
    }
    }

    struct TrainingData: Codable {
    var oneRepMax: [String: Double]?
    var workoutHistory: [WorkoutSession]?
    var oneRepMaxCount: [String: Int]?

    enum CodingKeys: String, CodingKey {
        case oneRepMax
        case workoutHistory
        case oneRepMaxCount
    }
```

```
    init(oneRepMax: [String : Double]? = [:], workoutHistory: [WorkoutSession]? = [],
oneRepMaxCount: [String : Int]? = [:]) {
        self.oneRepMax = oneRepMax
        self.workoutHistory = workoutHistory
        self.oneRepMaxCount = oneRepMaxCount
    }
}

protocol AnyWorkout {
    var id: UUID { get }
    var name: String { get }
    var isComplete: Bool { get set }
    var description: String? { get }
    var workoutType: WorkoutType? { get }
    var startDate: Date? { get }
    var endDate: Date? { get }
}

struct StrengthWorkout: Hashable, Identifiable, Codable, AnyWorkout {
    let id: UUID
    let name: String
    var isComplete: Bool
    let description: String?
    let exercises: [Exercise]?
    let supersets: [Superset]?
    var startDate: Date?
    var endDate: Date?
    let workoutType: WorkoutType?

    init(id: UUID, name: String, description: String, exercises: [Exercise], supersets: [Superset] =
[], startDate: Date?, endDate: Date?, isComplete: Bool, workoutType: WorkoutType?) {
        self.id = id
        self.name = name
        self.description = description
        self.exercises = exercises
        self.supersets = supersets
        self.startDate = startDate
        self.endDate = endDate
        self.isComplete = isComplete
        self.workoutType = workoutType
    }

    enum CodingKeys: CodingKey {
        case id
        case name
        case description
        case exercises
        case supersets
        case startDate
        case endDate
        case isComplete
        case workoutType
    }
```

```
}

struct Superset: Identifiable, Codable, Equatable, Hashable {
    let id: UUID
    let firstExercise: Exercise
    let secondExercise: Exercise
}

struct EnduranceWorkout: Hashable, Identifiable, Codable, AnyWorkout {
    let id: UUID
    let name: String
    var isComplete: Bool
    var startDate: Date?
    var endDate: Date?
    let duration: Duration?
    let instructions: String?
    let description: String?
    var perceivedExertion: PerceivedExertion?
    let activityType: ActivityType?
    let intensity: Intensity?
    let workoutType: WorkoutType?

    init(startDate: Date? = nil, endDate: Date? = nil, id: UUID, name: String, duration: Duration?,
instructions: String?, description: String?, perceivedExertion: PerceivedExertion? = nil,
isComplete: Bool, activityType: ActivityType?, intensity: Intensity?, workoutType:
WorkoutType?) {
        self.startDate = startDate
        self.endDate = endDate
        self.id = id
        self.name = name
        self.duration = duration
        self.instructions = instructions
        self.description = description
        self.perceivedExertion = perceivedExertion
        self.isComplete = isComplete
        self.activityType = activityType
        self.intensity = intensity
        self.workoutType = workoutType
    }

    }

    struct WorkoutSession: Identifiable, Codable {
    var exerciseSets: [ExerciseSet]
    var isComplete: Bool
    var currentExercise: Exercise?
    let id: UUID
    let date: Date
    var workout: AnyWorkout?

    init(date: Date = Date(), workout: AnyWorkout, exerciseSets: [ExerciseSet] = [], isComplete:
Bool = false) {
        self.id = UUID()
        self.date = date
```

```
        self.workout = workout
        self.exerciseSets = exerciseSets
        self.isComplete = isComplete
        if let workout = workout as? StrengthWorkout {
            self.currentExercise = workout.exercises?[0]
        } else {
            self.currentExercise = nil
        }
    }

    struct TrainingCalendar: Codable {
    let id: UUID
    let goal: TrainingGoal
    var trainingBlockIndex: Int = 0
    var trainingWeekIndex: Int = 0
    var trainingBlocks: [TrainingBlock]

    // Initialize the TrainingCalendar with a goal and create the first detailed training block.
    init(goal: TrainingGoal, id: UUID) {
        self.goal = goal
        self.trainingBlocks = []
        self.id = id
    }

    var currentTrainingBlock: TrainingBlock? {
        guard trainingBlockIndex < trainingBlocks.count else { return nil }
        return trainingBlocks[trainingBlockIndex]
    }

    var currentTrainingWeek: TrainingWeek? {
        guard let currentTrainingBlock, trainingWeekIndex <
currentTrainingBlock.trainingWeeks.count else { return nil }
        return currentTrainingBlock.trainingWeeks[trainingWeekIndex]
    }
}

struct TrainingBlock: Identifiable, Codable {
    let id: UUID
    let startDate: Date
    let endDate: Date
    let phase: Phase
    var trainingWeeks: [TrainingWeek]
    var isCurrent: Bool = false

    var duration: Int {
        let calendar = Calendar.current
        let components = calendar.dateComponents([.day], from: startDate, to: endDate)
        return components.day ?? 0
    }

    var daysRemaining: Int {
        let calendar = Calendar.current
        let currentDate = Date()
        let components = calendar.dateComponents([.day], from: currentDate, to: endDate)
```

```
        return components.day ?? 0
    }

    init(id: UUID, startDate: Date, endDate: Date, phase: Phase, trainingWeeks: [TrainingWeek]) {
        self.id = id
        self.startDate = startDate
        self.endDate = endDate
        self.phase = phase
        self.trainingWeeks = trainingWeeks
    }
    }

    struct TrainingWeek: Identifiable, Codable {
    let id: UUID
    let weekNumber: Int
    var workouts: [AnyWorkout]
    let startDate: Date
    let endDate: Date
    var isCurrent: Bool = false

    init(id: UUID = UUID(), weekNumber: Int, workouts: [AnyWorkout], startDate: Date, endDate:
Date) {
        self.id = id
        self.weekNumber = weekNumber
        self.workouts = workouts
        self.startDate = startDate
        self.endDate = endDate
    }
    }

    struct ExerciseSet: Codable, Identifiable {
    let id: UUID
    let exerciseName: String
    let reps: Int
    let weight: Double
    let date: Date
    let perceivedExertion: PerceivedExertion?

    init(exerciseName: String, reps: Int, weight: Double, date: Date, perceivedExertion:
PerceivedExertion?) {
        self.id = UUID()
        self.exerciseName = exerciseName
        self.reps = reps
        self.weight = weight
        self.date = date
        self.perceivedExertion = perceivedExertion
    }
    }
```