```
Powerlift/
├── PowerliftProApp.swift
├── Models/
│   ├── UserProfile.swift
│   ├── ExerciseType.swift
│   ├── Workout.swift
│   ├── WorkoutSet.swift
│   ├── WorkoutPlan.swift
│   └── AthleteLevel.swift
├── Managers/
│   ├── DataManager.swift
│   ├── VideoRecordingManager.swift
│   ├── CameraManager.swift
│   └── GoogleSheetsSync.swift
├── ViewModels/
│   └── WorkoutViewModel.swift
├── Views/
│   ├── Onboarding/
│   │   └── OnboardingView.swift
│   ├── Main/
│   │   └── MainView.swift
│   ├── Dashboard/
│   │   └── TodayWorkoutCard.swift
│   ├── Workout/
│   │   ├── WorkoutPlannerView.swift
│   │   ├── ExerciseDetailView.swift
│   │   └── ManualWorkoutCreatorView.swift
│   ├── Profile/
│   │   └── ProfileEditView.swift
│   └── Camera/
│   ├── CameraView.swift
│   ├── CameraRecordingView.swift
│   ├── VideoPlayerView.swift
│   ├── VideoPicker.swift
│   ├── VideoAnalysisView.swift
│   └── VideoTrimmerView.swift
└── Utilities/
├── AppColors.swift
├── ColorExtension.swift
├── BarbellTracker.swift
└── ProgressionAlgorithm.swift
```

📄 FILE CONTENTS

```
=============================================
FILE: PowerliftProApp.swift
PATH: Powerlift/
=============================================
swift
import SwiftUI

@main
struct PowerliftProApp: App {
var body: some Scene {
WindowGroup {
OnboardingView()
}
}
}
=============================================
FILE: UserProfile.swift
PATH: Powerlift/Models/
=============================================
swift
import Foundation

struct UserProfile: Codable {
var name: String
var age: Int
var weight: Double
var heightCm: Double
var gender: Gender
var athleteLevel: AthleteLevel
var goals: [String]
var squat1RM: Double?
var bench1RM: Double?
var deadlift1RM: Double?

enum Gender: String, Codable {
    case male = "Male"
    case female = "Female"
    case other = "Other"
}

static var mock: UserProfile {
    UserProfile(
        name: "Mario Rossi",
        age: 28,
        weight: 80.0,
        heightCm: 180.0,
        gender: .male,
        athleteLevel: .intermediate,
        goals: ["Increase Strength", "Build Muscle"],
        squat1RM: 140.0,
        bench1RM: 100.0,
        deadlift1RM: 180.0
    )
}
```

}

FILE: ExerciseType.swift

PATH: Powerlift/Models/

---

swift
import SwiftUI

enum ExerciseType: String, Codable, CaseIterable {
case squat = "Squat"
case benchPress = "Bench Press"
case deadlift = "Deadlift"

```
var icon: String {
    switch self {
    case .squat: return "figure.strengthtraining.traditional"
    case .benchPress: return "figure.cooldown"
    case .deadlift: return "figure.stand"
    }
}

var color: Color {
    switch self {
    case .squat: return .blue
    case .benchPress: return .green
    case .deadlift: return .red
    }
}
```

}

FILE: Workout.swift

PATH: Powerlift/Models/

```
swift
import Foundation

struct Workout: Identifiable, Codable {
var id = UUID()
var date: Date
var exerciseType: ExerciseType
var sets: [WorkoutSet]
var notes: String?
var videoURLs: [String]?
```

```
var totalVolume: Double {
    sets.reduce(0) { $0 + ($1.weight * Double($1.reps)) }
}

var topSet: WorkoutSet? {
    sets.max { $0.weight < $1.weight }
}

var averageRPE: Double {
    let total = sets.reduce(0.0) { $0 + $1.rpe }
    return sets.isEmpty ? 0 : total / Double(sets.count)
}
```

}

FILE: WorkoutSet.swift

PATH: Powerlift/Models/

---

swift

```
import Foundation

struct WorkoutSet: Identifiable, Codable {
var id = UUID()
var setNumber: Int
var weight: Double
var reps: Int
var rpe: Double
var isWarmup: Bool
var restTime: Int?
var videoURL: String?

var estimatedOneRM: Double {
    // Epley Formula
    weight * (1 + Double(reps) / 30.0)
}

var intensity: Double {
    guard let maxWeight = calculateMaxWeight() else { return 0 }
    return (weight / maxWeight) * 100
}

private func calculateMaxWeight() -> Double? {
    // This should reference actual 1RM from profile
    return estimatedOneRM
}
```

}

FILE: WorkoutPlan.swift

PATH: Powerlift/Models/

---

swift

```
import Foundation

struct WorkoutPlan: Identifiable, Codable {
var id = UUID()
var name: String
var duration: Int // weeks
var workoutsPerWeek: Int
var exercises: [ExerciseType]
var startDate: Date
var endDate: Date

var weeksRemaining: Int {
    let calendar = Calendar.current
    let weeks = calendar.dateComponents([.weekOfYear], from: Date(), to: endDate)
    return max(0, weeks.weekOfYear ?? 0)
}

var progress: Double {
    let calendar = Calendar.current
    let totalDays = calendar.dateComponents([.day], from: startDate, to: endDate).day ?? 1
    let elapsedDays = calendar.dateComponents([.day], from: startDate, to: Date()).day ?? 0
    return min(1.0, Double(elapsedDays) / Double(totalDays))
}

static var mock: WorkoutPlan {
    WorkoutPlan(
        name: "Beginner Strength",
        duration: 12,
        workoutsPerWeek: 3,
        exercises: [.squat, .benchPress, .deadlift],
        startDate: Date().addingTimeInterval(-30 * 24 * 60 * 60),
        endDate: Date().addingTimeInterval(60 * 24 * 60 * 60)
    )
}

}
```

FILE: AthleteLevel.swift

PATH: Powerlift/Models/

swift

```
import Foundation

enum AthleteLevel: String, Codable, CaseIterable {
case beginner = "Beginner"
case intermediate = "Intermediate"
case advanced = "Advanced"
case elite = "Elite"

var description: String {
    switch self {
    case .beginner:
        return "0-1 years of consistent training"
    case .intermediate:
        return "1-3 years of consistent training"
    case .advanced:
        return "3-5 years of consistent training"
    case .elite:
        return "5+ years of consistent training"
    }
}

var weeklyProgressionRate: Double {
    switch self {
    case .beginner: return 0.025      // 2.5% per week
    case .intermediate: return 0.0125 // 1.25% per week
    case .advanced: return 0.00625    // 0.625% per week
    case .elite: return 0.003125      // 0.3125% per week
    }
}

var deloadFrequency: Int {
    switch self {
    case .beginner: return 6      // every 6 weeks
    case .intermediate: return 4  // every 4 weeks
    case .advanced: return 3      // every 3 weeks
    case .elite: return 3         // every 3 weeks
    }
}

var recommendedVolume: String {
    switch self {
    case .beginner: return "3-4 sets per exercise"
    case .intermediate: return "4-6 sets per exercise"
    case .advanced: return "5-8 sets per exercise"
    case .elite: return "6-10 sets per exercise"
    }
}
```

}

FILE: DataManager.swift

PATH: Powerlift/Managers/

---

swift
import Foundation

class DataManager: ObservableObject {
@Published var userProfile: UserProfile?
@Published var workouts: [Workout] = []
@Published var currentPlan: WorkoutPlan?

```
private let userDefaultsKey = "powerlift_data"

init() {
    loadData()
}

// MARK: - Save/Load

func saveData() {
    let data = AppData(
        userProfile: userProfile,
        workouts: workouts,
        currentPlan: currentPlan
    )

    if let encoded = try? JSONEncoder().encode(data) {
        UserDefaults.standard.set(encoded, forKey: userDefaultsKey)
    }
}

func loadData() {
    guard let data = UserDefaults.standard.data(forKey: userDefaultsKey),
          let decoded = try? JSONDecoder().decode(AppData.self, from:
data) else {
        return
    }

    userProfile = decoded.userProfile
    workouts = decoded.workouts
    currentPlan = decoded.currentPlan
}

// MARK: - Workouts

func addWorkout(_ workout: Workout) {
    workouts.append(workout)
    saveData()
}

func updateWorkout(_ workout: Workout) {
    if let index = workouts.firstIndex(where: { $0.id == workout.id }) {
        workouts[index] = workout
        saveData()
    }
}

func deleteWorkout(_ workout: Workout) {
    workouts.removeAll { $0.id == workout.id }
    saveData()
}

func workouts(for exercise: ExerciseType) -> [Workout] {
    workouts.filter { $0.exerciseType == exercise }
```

```
        .sorted { $0.date > $1.date }
}

func recentWorkouts(limit: Int = 10) -> [Workout] {
    Array(workouts.sorted { $0.date > $1.date }.prefix(limit))
}

// MARK: - Statistics

func topSet(for exercise: ExerciseType) -> WorkoutSet? {
    workouts(for: exercise)
        .compactMap { $0.topSet }
        .max { $0.weight < $1.weight }
}

func estimatedOneRM(for exercise: ExerciseType) -> Double {
    guard let topSet = topSet(for: exercise) else { return 0 }
    return topSet.estimatedOneRM
}

func totalVolume(for exercise: ExerciseType, in days: Int = 30) -> Double
{
    let cutoffDate = Calendar.current.date(byAdding: .day, value: -days,
to: Date()) ?? Date()
    return workouts(for: exercise)
        .filter { $0.date >= cutoffDate }
        .reduce(0) { $0 + $1.totalVolume }
}

func workoutCount(in days: Int = 30) -> Int {
    let cutoffDate = Calendar.current.date(byAdding: .day, value: -days,
to: Date()) ?? Date()
    return workouts.filter { $0.date >= cutoffDate }.count
}

// MARK: - Progress

func progressData(for exercise: ExerciseType, weeks: Int = 12) -> [(Date,
Double)] {
    let cutoffDate = Calendar.current.date(byAdding: .weekOfYear, value: -
weeks, to: Date()) ?? Date()
```

```
    return workouts(for: exercise)
        .filter { $0.date >= cutoffDate }
        .compactMap { workout -> (Date, Double)? in
            guard let topSet = workout.topSet else { return nil }
            return (workout.date, topSet.estimatedOneRM)
        }
        .sorted { $0.0 < $1.0 }
}

func volumeData(for exercise: ExerciseType, weeks: Int = 12) -> [(Date,
Double)] {
    let cutoffDate = Calendar.current.date(byAdding: .weekOfYear, value: -
weeks, to: Date()) ?? Date()

    let calendar = Calendar.current
    let workoutsInRange = workouts(for: exercise)
        .filter { $0.date >= cutoffDate }

    var weeklyVolume: [Date: Double] = [:]

    for workout in workoutsInRange {
        let weekStart = calendar.dateInterval(of: .weekOfYear, for:
workout.date)?.start ?? workout.date
        weeklyVolume[weekStart, default: 0] += workout.totalVolume
    }

    return weeklyVolume.sorted { $0.key < $1.key }
}

// MARK: - Profile

func updateProfile(_ profile: UserProfile) {
    userProfile = profile
    saveData()
}

func update1RM(for exercise: ExerciseType, value: Double) {
    guard var profile = userProfile else { return }

    switch exercise {
```

```
    case .squat:
        profile.squat1RM = value
    case .benchPress:
        profile.bench1RM = value
    case .deadlift:
        profile.deadlift1RM = value
    }

    updateProfile(profile)
}

}

// MARK: - App Data Model

struct AppData: Codable {
var userProfile: UserProfile?
var workouts: [Workout]
var currentPlan: WorkoutPlan?
}
================================================
FILE: VideoRecordingManager.swift
PATH: Powerlift/Managers/
================================================
swift
import AVFoundation
import SwiftUI

class VideoRecordingManager: NSObject, ObservableObject {
@Published var isRecording = false
@Published var recordedVideoURL: URL?
@Published var permissionGranted = false

private var captureSession: AVCaptureSession?
private var videoOutput: AVCaptureMovieFileOutput?
private var previewLayer: AVCaptureVideoPreviewLayer?

override init() {
    super.init()
}

func requestPermission() {
    AVCaptureDevice.requestAccess(for: .video) { [weak self] granted in
```

```
        DispatchQueue.main.async {
            self?.permissionGranted = granted
            if granted {
                self?.setupCamera()
            }
        }
    }
}

private func setupCamera() {
    captureSession = AVCaptureSession()
    captureSession?.sessionPreset = .high

    guard let videoCaptureDevice =
AVCaptureDevice.default(.builtInWideAngleCamera, for: .video, position:
.back),
          let videoInput = try? AVCaptureDeviceInput(device:
videoCaptureDevice),
          let captureSession = captureSession else {
        return
    }

    if captureSession.canAddInput(videoInput) {
        captureSession.addInput(videoInput)
    }

    videoOutput = AVCaptureMovieFileOutput()
    if let videoOutput = videoOutput,
captureSession.canAddOutput(videoOutput) {
        captureSession.addOutput(videoOutput)
    }

    DispatchQueue.global(qos: .userInitiated).async {
        captureSession.startRunning()
    }
}

func getPreviewLayer() -> AVCaptureVideoPreviewLayer? {
    guard let captureSession = captureSession else { return nil }

    if previewLayer == nil {
        previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
        previewLayer?.videoGravity = .resizeAspectFill
    }

    return previewLayer
}

func startRecording() {
    guard let videoOutput = videoOutput, !videoOutput.isRecording else {
return }

    let outputURL = FileManager.default.temporaryDirectory
        .appendingPathComponent(UUID().uuidString)
        .appendingPathExtension("mov")

    videoOutput.startRecording(to: outputURL, recordingDelegate: self)
    isRecording = true
}

func stopRecording() {
    videoOutput?.stopRecording()
    isRecording = false
}

func cleanup() {
    captureSession?.stopRunning()
    captureSession = nil
    videoOutput = nil
    previewLayer = nil
}
```

```
}

extension VideoRecordingManager: AVCaptureFileOutputRecordingDelegate {
func fileOutput(_ output: AVCaptureFileOutput, didFinishRecordingTo outputFileURL: URL, from
connections: [AVCaptureConnection], error: Error?) {
if error == nil {
DispatchQueue.main.async {
self.recordedVideoURL = outputFileURL
}
}
}
```

```
}
=============================================
FILE: CameraManager.swift
PATH: Powerlift/Managers/
=============================================
swift
import AVFoundation
import SwiftUI

class CameraManager: NSObject, ObservableObject {
@Published var isSessionRunning = false
@Published var capturedImage: UIImage?

private var captureSession: AVCaptureSession?
private var photoOutput: AVCapturePhotoOutput?
private var previewLayer: AVCaptureVideoPreviewLayer?

func checkAuthorization() {
    switch AVCaptureDevice.authorizationStatus(for: .video) {
    case .authorized:
        setupSession()
    case .notDetermined:
        AVCaptureDevice.requestAccess(for: .video) { [weak self] granted in
            if granted {
                DispatchQueue.main.async {
                    self?.setupSession()
                }
            }
        }
    default:
        break
    }
}

private func setupSession() {
    let session = AVCaptureSession()
    session.beginConfiguration()

    guard let videoDevice = AVCaptureDevice.default(.builtInWideAngleCamera, for: .video, position: .back),
          let videoDeviceInput = try? AVCaptureDeviceInput(device:
videoDevice),
          session.canAddInput(videoDeviceInput) else {
        return
    }

    session.addInput(videoDeviceInput)

    let photoOutput = AVCapturePhotoOutput()
    guard session.canAddOutput(photoOutput) else { return }

    session.addOutput(photoOutput)
    session.commitConfiguration()

    self.captureSession = session
    self.photoOutput = photoOutput

    DispatchQueue.global(qos: .userInitiated).async { [weak self] in
        self?.captureSession?.startRunning()
        DispatchQueue.main.async {
            self?.isSessionRunning = true
        }
    }
}

func getPreviewLayer() -> AVCaptureVideoPreviewLayer? {
    guard let captureSession = captureSession else { return nil }

    if previewLayer == nil {
        let layer = AVCaptureVideoPreviewLayer(session: captureSession)
        layer.videoGravity = .resizeAspectFill
        previewLayer = layer
    }

    return previewLayer
}

func capturePhoto() {
    guard let photoOutput = photoOutput else { return }

    let settings = AVCapturePhotoSettings()
    photoOutput.capturePhoto(with: settings, delegate: self)
}

func stopSession() {
    captureSession?.stopRunning()
    isSessionRunning = false
}
```

}

extension CameraManager: AVCapturePhotoCaptureDelegate {
func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto photo: AVCapturePhoto, error: Error?) {
guard let imageData = photo.fileDataRepresentation(),
let image = UIImage(data: imageData) else {
return
}

```
    DispatchQueue.main.async {
        self.capturedImage = image
    }
}
```

}

FILE: GoogleSheetsSync.swift

PATH: Powerlift/Managers/

---

swift
import Foundation

class GoogleSheetsSync {
// Placeholder for Google Sheets integration
// Requires Google Sheets API setup

```
func exportWorkouts(_ workouts: [Workout]) {
    // TODO: Implement Google Sheets export
    print("Exporting \(workouts.count) workouts to Google Sheets")
}
```

```
func importWorkouts() -> [Workout] {
    // TODO: Implement Google Sheets import
    print("Importing workouts from Google Sheets")
    return []
}

func syncWithCloud() {
    // TODO: Implement cloud sync
    print("Syncing with cloud")
}
```

}

FILE: WorkoutViewModel.swift

PATH: Powerlift/ViewModels/

---

swift
import SwiftUI

class WorkoutViewModel: ObservableObject {
@Published var currentWorkout: Workout?
@Published var currentSets: [WorkoutSet] = []
@Published var isActiveWorkout = false

```
let dataManager: DataManager

init(dataManager: DataManager) {
    self.dataManager = dataManager
}

func startWorkout(exercise: ExerciseType) {
    currentWorkout = Workout(
        date: Date(),
        exerciseType: exercise,
        sets: []
    )
    currentSets = []
    isActiveWorkout = true
}

func addSet(weight: Double, reps: Int, rpe: Double, isWarmup: Bool =
false) {
    let set = WorkoutSet(
        setNumber: currentSets.count + 1,
        weight: weight,
        reps: reps,
        rpe: rpe,
        isWarmup: isWarmup
    )
    currentSets.append(set)
}

func removeLastSet() {
    if !currentSets.isEmpty {
        currentSets.removeLast()
    }
}

func finishWorkout(notes: String? = nil) {
    guard var workout = currentWorkout else { return }
    workout.sets = currentSets
    workout.notes = notes

    dataManager.addWorkout(workout)

    currentWorkout = nil
    currentSets = []
    isActiveWorkout = false
}

func cancelWorkout() {
    currentWorkout = nil
    currentSets = []
    isActiveWorkout = false
}

func getSuggestedWeight(for exercise: ExerciseType, targetReps: Int,
targetRPE: Double) -> Double {
    let oneRM = dataManager.estimatedOneRM(for: exercise)
    guard oneRM > 0 else { return 0 }

    let intensity = ProgressionAlgorithm.calculateIntensity(reps:
targetReps, rpe: targetRPE)
    return oneRM * intensity
}
```

}

FILE: AppColors.swift

PATH: Powerlift/Utilities/

---

swift

```
import SwiftUI

struct AppColors {
// Primary Colors
static let primary = Color(red: 0.83, green: 0.13, blue: 0.18) // Cherry Red #D42129
static let primaryLight = Color(red: 0.90, green: 0.25, blue: 0.30)
static let primaryDark = Color(red: 0.70, green: 0.10, blue: 0.15)
```

```
// Background Colors
static let background = Color(red: 0.09, green: 0.09, blue: 0.09) //
#171717
static let cardBackground = Color(red: 0.12, green: 0.12, blue: 0.12) //
#1F1F1F
static let cardBackgroundLight = Color(red: 0.15, green: 0.15, blue: 0.15)
// #262626

// Text Colors
static let textPrimary = Color.white
static let textSecondary = Color(red: 0.7, green: 0.7, blue: 0.7)
static let textTertiary = Color(red: 0.5, green: 0.5, blue: 0.5)

// Accent Colors
static let success = Color(red: 0.20, green: 0.78, blue: 0.35) // Green
static let warning = Color(red: 1.0, green: 0.80, blue: 0.0) // Yellow
static let error = Color(red: 0.96, green: 0.26, blue: 0.21) // Red
static let info = Color(red: 0.20, green: 0.60, blue: 0.86) // Blue

// Exercise Colors
static let squatColor = Color.blue
static let benchColor = Color.green
static let deadliftColor = Color.orange

// Gradient
static let primaryGradient = LinearGradient(
    colors: [primary, primaryDark],
    startPoint: .topLeading,
    endPoint: .bottomTrailing
)

static let backgroundGradient = LinearGradient(
    colors: [background, cardBackground],
    startPoint: .top,
    endPoint: .bottom
)

// RPE Colors
static func rpeColor(for rpe: Double) -> Color {
    switch rpe {
    case 0..<6:
        return .green
    case 6..<8:
        return .yellow
    case 8..<9:
        return .orange
    default:
        return .red
    }
}

// Intensity Colors
static func intensityColor(for intensity: Double) -> Color {
    switch intensity {
    case 0..<60:
        return .blue
    case 60..<75:
        return .green
    case 75..<85:
        return .yellow
    case 85..<90:
        return .orange
    default:
        return .red
    }
}

}

// MARK: - Extensions

extension Color {
init(hex: String) {
let hex = hex.trimmingCharacters(in: CharacterSet.alphanumerics.inverted)
var int: UInt64 = 0
Scanner(string: hex).scanHexInt64(&int)
let a, r, g, b: UInt64
switch hex.count {
case 3: // RGB (12-bit)
(a, r, g, b) = (255, (int >> 8) * 17, (int >> 4 & 0xF) * 17, (int & 0xF) * 17)
case 6: // RGB (24-bit)
(a, r, g, b) = (255, int >> 16, int >> 8 & 0xFF, int & 0xFF)
case 8: // ARGB (32-bit)
(a, r, g, b) = (int >> 24, int >> 16 & 0xFF, int >> 8 & 0xFF, int & 0xFF)
default:
(a, r, g, b) = (255, 0, 0, 0)
}

    self.init(
        .sRGB,
        red: Double(r) / 255,
        green: Double(g) / 255,
        blue:  Double(b) / 255,
        opacity: Double(a) / 255
    )
}

}
```

FILE: ColorExtension.swift

PATH: Powerlift/Utilities/

```
swift
import SwiftUI

extension Color {
static let theme = ColorTheme()
}

struct ColorTheme {
let accent = Color("AccentColor")
let background = Color("BackgroundColor")
let secondaryBackground = Color("SecondaryBackgroundColor")
let text = Color("TextColor")
let secondaryText = Color("SecondaryTextColor")

// Exercise specific colors
let squat = Color.blue
let bench = Color.green
let deadlift = Color.red

// Status colors
let success = Color.green
let warning = Color.orange
let error = Color.red
let info = Color.blue

}

extension Color {
func lighter(by percentage: CGFloat = 0.2) -> Color {
return self.adjust(by: abs(percentage))
}

func darker(by percentage: CGFloat = 0.2) -> Color {
    return self.adjust(by: -abs(percentage))
}

func adjust(by percentage: CGFloat) -> Color {
    var red: CGFloat = 0, green: CGFloat = 0, blue: CGFloat = 0, alpha:
```

```
CGFloat = 0

    UIColor(self).getRed(&red, green: &green, blue: &blue, alpha: &alpha)

    return Color(
        red: min(red + percentage, 1.0),
        green: min(green + percentage, 1.0),
        blue: min(blue + percentage, 1.0),
        opacity: alpha
    )
}
```

}

FILE: BarbellTracker.swift

PATH: Powerlift/Utilities/

```
swift
import AVFoundation
import Vision
import CoreML

class BarbellTracker {
// Placeholder for video analysis/barbell tracking
// Would use Vision framework + CoreML
```

```
func analyzeVideo(url: URL, completion: @escaping (VideoAnalysisResult) ->
Void) {
    // TODO: Implement actual video analysis

    // Simulate analysis delay
    DispatchQueue.main.asyncAfter(deadline: .now() + 2.0) {
        let mockResult = VideoAnalysisResult(
            repCount: 5,
            averageVelocity: 0.75,
            peakVelocity: 1.2,
            rangeOfMotion: 45.5,
            barPath: [],
            duration: 12.5
        )
        completion(mockResult)
    }
}

func trackBarPosition(in videoURL: URL) -> [CGPoint] {
    // TODO: Implement barbell position tracking
    return []
}

func detectRepetitions(from positions: [CGPoint]) -> Int {
    // TODO: Implement rep detection from position data
    return 0
}

func calculateVelocity(from positions: [CGPoint], frameRate: Double) ->
[Double] {
    // TODO: Calculate velocity from position changes
    return []
}

}

struct VideoAnalysisResult {
let repCount: Int
let averageVelocity: Double
let peakVelocity: Double
let rangeOfMotion: Double
let barPath: [CGPoint]
let duration: Double
}
================================================
FILE: ProgressionAlgorithm.swift
PATH: Powerlift/Utilities/
================================================
swift
import Foundation

struct ProgressionAlgorithm {

// MARK: - Calculate Next Workout Weight
```

```
static func calculateNextWeight(
    currentWeight: Double,
    lastRPE: Double,
    athleteLevel: AthleteLevel,
    targetRPE: Double = 8.0
) -> Double {
    let rpeDifference = targetRPE - lastRPE
    let progressionRate = athleteLevel.weeklyProgressionRate

    var weightIncrease: Double = 0

    if rpeDifference < -1 {
        // Last set was too hard, decrease weight
        weightIncrease = currentWeight * -0.05
    } else if rpeDifference > 1 {
        // Last set was too easy, increase weight more
        weightIncrease = currentWeight * (progressionRate * 2)
    } else {
        // Normal progression
        weightIncrease = currentWeight * progressionRate
    }

    // Round to nearest 2.5kg
    let newWeight = currentWeight + weightIncrease
    return round(newWeight / 2.5) * 2.5
}

// MARK: - Calculate Training Intensity

static func calculateIntensity(reps: Int, rpe: Double) -> Double {
    // Estimate percentage of 1RM based on reps and RPE
    // Using Epley formula adjusted for RPE

    let baseIntensity = 1.0 / (1.0 + Double(reps) / 30.0)
    let rpeAdjustment = (10 - rpe) * 0.025

    return max(0.5, min(1.0, baseIntensity - rpeAdjustment))
}

// MARK: - Recommend Sets and Reps
```

```swift
static func recommendVolume(
    exercise: ExerciseType,
    athleteLevel: AthleteLevel,
    trainingPhase: TrainingPhase
) -> (sets: Int, reps: ClosedRange<Int>) {

    switch trainingPhase {
    case .hypertrophy:
        switch athleteLevel {
        case .beginner:
            return (3, 8...12)
        case .intermediate:
            return (4, 8...12)
        case .advanced, .elite:
            return (5, 6...12)
        }

    case .strength:
        switch athleteLevel {
        case .beginner:
            return (3, 4...6)
        case .intermediate:
            return (4, 3...5)
        case .advanced, .elite:
            return (5, 1...5)
        }

    case .power:
        switch athleteLevel {
        case .beginner:
            return (3, 3...5)
        case .intermediate:
            return (4, 2...4)
        case .advanced, .elite:
            return (5, 1...3)
        }

    case .deload:
        return (2, 5...8)
    }
}
```

```
// MARK: - Calculate Estimated 1RM

static func estimateOneRM(weight: Double, reps: Int, rpe: Double) ->
Double {
    // Epley Formula with RPE adjustment
    let baseEstimate = weight * (1 + Double(reps) / 30.0)
    let rpeMultiplier = 1.0 + ((10 - rpe) * 0.025)

    return baseEstimate * rpeMultiplier
}

// MARK: - Autoregulation

static func adjustForFatigue(
    plannedWeight: Double,
    recentAverageRPE: Double,
    targetRPE: Double = 8.0
) -> Double {

    if recentAverageRPE > 9.0 {
        // High fatigue, reduce weight significantly
        return plannedWeight * 0.90
    } else if recentAverageRPE > 8.5 {
        // Moderate fatigue, reduce slightly
        return plannedWeight * 0.95
    } else if recentAverageRPE < 7.0 {
        // Low fatigue, can push more
        return plannedWeight * 1.05
    }

    return plannedWeight
}

// MARK: - Deload Check

static func shouldDeload(
    weekNumber: Int,
    athleteLevel: AthleteLevel,
    averageRPE: Double
) -> Bool {
    let deloadFrequency = athleteLevel.deloadFrequency

    // Check if it's time for scheduled deload
    if weekNumber % deloadFrequency == 0 {
        return true
    }

    // Check if athlete is consistently hitting high RPEs
    if averageRPE >= 9.5 {
        return true
    }

    return false
}
```

```
}

// MARK: - Supporting Types

enum TrainingPhase {
case hypertrophy
case strength
case power
case deload
}
```

==
FILE: OnboardingView.swift
PATH: Powerlift/Views/Onboarding/
==
swift

```
import SwiftUI

struct OnboardingView: View {
@StateObject private var dataManager = DataManager()
@State private var currentStep = 0
@State private var name = ""
@State private var age = ""
@State private var weight = ""
@State private var height = ""
@State private var selectedGender: UserProfile.Gender = .male
@State private var selectedLevel: AthleteLevel = .beginner
```

```
@State private var selectedGoals: Set = [ ]
@State private var showMainView = false

let goals = ["Increase Strength", "Build Muscle", "Lose Fat", "Improve Technique", "Compete in Powerlifting"]

var body: some View {
    ZStack {
        AppColors.background
            .ignoresSafeArea()

        if showMainView {
            MainView()
                .environmentObject(dataManager)
        } else {
            VStack(spacing: 0) {
                // Progress Bar
                progressBar

                // Content
                ScrollView {
                    VStack(spacing: 24) {
                        switch currentStep {
                        case 0:
                            welcomeStep
                        case 1:
                            personalInfoStep
                        case 2:
                            levelStep
                        case 3:
                            goalsStep
                        case 4:
                            maxesStep
                        default:
                            EmptyView()
                        }
                    }
                    .padding()
                }

                // Navigation Buttons
                navigationButtons
            }
        }
    }
}

// MARK: - Progress Bar

private var progressBar: some View {
    GeometryReader { geometry in
        ZStack(alignment: .leading) {
            Rectangle()
                .fill(AppColors.cardBackground)
                .frame(height: 4)

            Rectangle()
                .fill(AppColors.primary)
                .frame(width: geometry.size.width * CGFloat(currentStep + 1) / 5, height: 4)
        }
    }
    .frame(height: 4)
}

// MARK: - Welcome Step

private var welcomeStep: some View {
    VStack(spacing: 32) {
        Spacer()

        Image(systemName: "figure.strengthtraining.traditional")
            .font(.system(size: 100))
            .foregroundColor(AppColors.primary)

        Text("Welcome to Powerlift")
            .font(.system(size: 36, weight: .bold))
            .foregroundColor(AppColors.textPrimary)
            .multilineTextAlignment(.center)

        Text("Your AI-powered powerlifting coach")
            .font(.system(size: 18))
            .foregroundColor(AppColors.textSecondary)
            .multilineTextAlignment(.center)

        Spacer()
    }
}

// MARK: - Personal Info Step

private var personalInfoStep: some View {
    VStack(alignment: .leading, spacing: 24) {
        Text("Tell us about yourself")
            .font(.system(size: 28, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        VStack(alignment: .leading, spacing: 8) {
            Text("Name")
                .font(.system(size: 14, weight: .semibold))
                .foregroundColor(AppColors.textSecondary)

            TextField("Enter your name", text: $name)
                .textFieldStyle(CustomTextFieldStyle())
        }

        HStack(spacing: 16) {
            VStack(alignment: .leading, spacing: 8) {
                Text("Age")
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textSecondary)

                TextField("Age", text: $age)
                    .keyboardType(.numberPad)
                    .textFieldStyle(CustomTextFieldStyle())
            }

            VStack(alignment: .leading, spacing: 8) {
                Text("Weight (kg)")
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textSecondary)

                TextField("Weight", text: $weight)
                    .keyboardType(.decimalPad)
                    .textFieldStyle(CustomTextFieldStyle())
            }
        }

        VStack(alignment: .leading, spacing: 8) {
            Text("Height (cm)")
                .font(.system(size: 14, weight: .semibold))
                .foregroundColor(AppColors.textSecondary)

            TextField("Height", text: $height)
                .keyboardType(.numberPad)
                .textFieldStyle(CustomTextFieldStyle())
        }

        VStack(alignment: .leading, spacing: 8) {
            Text("Gender")
                .font(.system(size: 14, weight: .semibold))
                .foregroundColor(AppColors.textSecondary)

            Picker("Gender", selection: $selectedGender) {
                Text("Male").tag(UserProfile.Gender.male)
                Text("Female").tag(UserProfile.Gender.female)
                Text("Other").tag(UserProfile.Gender.other)
            }
            .pickerStyle(SegmentedPickerStyle())
        }

        Spacer()
    }
}

// MARK: - Level Step

private var levelStep: some View {
    VStack(alignment: .leading, spacing: 24) {
        Text("What's your experience level?")
            .font(.system(size: 28, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        ForEach(AthleteLevel.allCases, id: \.self) { level in
            Button(action: {
```

```
                selectedLevel = level
            }) {
                HStack {
                    VStack(alignment: .leading, spacing: 8) {
                        Text(level.rawValue)
                            .font(.system(size: 18, weight: .semibold))
                            .foregroundColor(AppColors.textPrimary)

                        Text(level.description)
                            .font(.system(size: 14))
                            .foregroundColor(AppColors.textSecondary)
                    }

                    Spacer()

                    if selectedLevel == level {
                        Image(systemName: "checkmark.circle.fill")
                            .foregroundColor(AppColors.primary)
                            .font(.system(size: 24))
                    }
                }
                .padding()
                .background(
                    RoundedRectangle(cornerRadius: 12)
                        .fill(selectedLevel == level ? 
AppColors.primary.opacity(0.1) : AppColors.cardBackground)
                )
                .overlay(
                    RoundedRectangle(cornerRadius: 12)
                        .stroke(selectedLevel == level ? AppColors.primary 
: Color.clear, lineWidth: 2)
                )
            }
        }

        Spacer()
    }
}

// MARK: - Goals Step
```

```
private var goalsStep: some View {
    VStack(alignment: .leading, spacing: 24) {
        Text("What are your goals?")
            .font(.system(size: 28, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        Text("Select all that apply")
            .font(.system(size: 16))
            .foregroundColor(AppColors.textSecondary)

        ForEach(goals, id: \.self) { goal in
            Button(action: {
                if selectedGoals.contains(goal) {
                    selectedGoals.remove(goal)
                } else {
                    selectedGoals.insert(goal)
                }
            }) {
                HStack {
                    Text(goal)
                        .font(.system(size: 16))
                        .foregroundColor(AppColors.textPrimary)

                    Spacer()

                    if selectedGoals.contains(goal) {
                        Image(systemName: "checkmark.circle.fill")
                            .foregroundColor(AppColors.primary)
                            .font(.system(size: 24))
                    } else {
                        Image(systemName: "circle")
                            .foregroundColor(AppColors.textSecondary)
                            .font(.system(size: 24))
                    }
                }
                .padding()
                .background(
                    RoundedRectangle(cornerRadius: 12)
                        .fill(selectedGoals.contains(goal) ?
AppColors.primary.opacity(0.1) : AppColors.cardBackground)
                )
                .overlay(
                    RoundedRectangle(cornerRadius: 12)
                        .stroke(selectedGoals.contains(goal) ?
AppColors.primary : Color.clear, lineWidth: 2)
                )
            }
        }

        Spacer()
    }
}

// MARK: - Maxes Step

private var maxesStep: some View {
    VStack(alignment: .leading, spacing: 24) {
        Text("Current Maxes (Optional)")
            .font(.system(size: 28, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        Text("Enter your estimated 1RM if you know them")
            .font(.system(size: 16))
            .foregroundColor(AppColors.textSecondary)

        // Max inputs will be added here

        Spacer()
    }
}

// MARK: - Navigation Buttons

private var navigationButtons: some View {
    HStack(spacing: 16) {
        if currentStep > 0 {
            Button(action: {
                withAnimation {
                    currentStep -= 1
                }
            }) {
                Text("Back")
```

```
                    .font(.system(size: 16, weight: .semibold))
                    .foregroundColor(AppColors.textSecondary)
                    .frame(maxWidth: .infinity)
                    .padding()
                    .background(AppColors.cardBackground)
                    .cornerRadius(12)
            }
        }

        Button(action: {
            if currentStep < 4 {
                withAnimation {
                    currentStep += 1
                }
            } else {
                completeOnboarding()
            }
        }) {
            Text(currentStep < 4 ? "Next" : "Get Started")
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding()
                .background(AppColors.primary)
                .cornerRadius(12)
        }
    }
    .padding()
}

// MARK: - Complete Onboarding

private func completeOnboarding() {
    let profile = UserProfile(
        name: name,
        age: Int(age) ?? 25,
        weight: Double(weight) ?? 75.0,
        heightCm: Double(height) ?? 175.0,
        gender: selectedGender,
        athleteLevel: selectedLevel,
        goals: Array(selectedGoals),
        squat1RM: nil,
        bench1RM: nil,
        deadlift1RM: nil
    )

    dataManager.updateProfile(profile)

    withAnimation {
        showMainView = true
    }
}

}

// MARK: - Custom Text Field Style

struct CustomTextFieldStyle: TextFieldStyle {
func _body(configuration: TextField<Self._Label>) -> some View {
configuration
.padding()
.background(AppColors.cardBackground)
.cornerRadius(8)
.foregroundColor(AppColors.textPrimary)
}
}
============================================
FILE: MainView.swift
PATH: Powerlift/Views/Main/
============================================
swift
import SwiftUI

struct MainView: View {
@EnvironmentObject var dataManager: DataManager
@State private var selectedTab = 0

var body: some View {
    TabView(selection: $selectedTab) {
        DashboardView()
            .environmentObject(dataManager)
            .tabItem {
                Label("Home", systemImage: "house.fill")
            }
            .tag(0)

        WorkoutPlannerView()
            .environmentObject(dataManager)
            .tabItem {
                Label("Workout", systemImage:
"figure.strengthtraining.traditional")
            }
            .tag(1)

        CameraRecordingView()
            .tabItem {
                Label("Record", systemImage: "video.fill")
            }
            .tag(2)

        ProgressView()
            .environmentObject(dataManager)
            .tabItem {
                Label("Progress", systemImage:
"chart.line.uptrend.xyaxis")
            }
            .tag(3)

        ProfileView()
            .environmentObject(dataManager)
            .tabItem {
                Label("Profile", systemImage: "person.fill")
            }
            .tag(4)
    }
    .accentColor(AppColors.primary)
}

}

// MARK: - Dashboard View

struct DashboardView: View {
@EnvironmentObject var dataManager: DataManager

var body: some View {
    NavigationView {
        ScrollView {
            VStack(spacing: 20) {
                TodayWorkoutCard()
                    .environmentObject(dataManager)

                statsSection

                recentWorkoutsSection
            }
            .padding()
        }
        .navigationTitle("Dashboard")
        .background(AppColors.background.ignoresSafeArea())
    }
}

private var statsSection: some View {
    VStack(alignment: .leading, spacing: 16) {
        Text("Your Stats")
            .font(.system(size: 20, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        HStack(spacing: 12) {
            StatCard(title: "Squat", value: "\
(Int(dataManager.estimatedOneRM(for: .squat)))kg", icon:
"figure.strengthtraining.traditional", color: .blue)
            StatCard(title: "Bench", value: "\
(Int(dataManager.estimatedOneRM(for: .benchPress)))kg", icon:
"figure.cooldown", color: .green)
            StatCard(title: "Deadlift", value: "\
(Int(dataManager.estimatedOneRM(for: .deadlift)))kg", icon:
"figure.stand", color: .red)
        }
    }
}

private var recentWorkoutsSection: some View {
    VStack(alignment: .leading, spacing: 16) {
        Text("Recent Workouts")
            .font(.system(size: 20, weight: .bold))
            .foregroundColor(AppColors.textPrimary)
        ForEach(dataManager.recentWorkouts(limit: 5)) { workout in
            WorkoutRowView(workout: workout)
        }
    }
}
```

}

// MARK: - Progress View

struct ProgressView: View {
@EnvironmentObject var dataManager: DataManager
@State private var selectedExercise: ExerciseType = .squat

```
var body: some View {
    NavigationView {
        ScrollView {
            VStack(spacing: 20) {
                exercisePicker

                oneRMCard

                volumeCard
            }
            .padding()
        }
        .navigationTitle("Progress")
        .background(AppColors.background.ignoresSafeArea())
    }
}

private var exercisePicker: some View {
    Picker("Exercise", selection: $selectedExercise) {
        ForEach(ExerciseType.allCases, id: \.self) { exercise in
            Text(exercise.rawValue).tag(exercise)
        }
    }
    .pickerStyle(SegmentedPickerStyle())
}

private var oneRMCard: some View {
    VStack(alignment: .leading, spacing: 12) {
        Text("Estimated 1RM Progress")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        Text("\(Int(dataManager.estimatedOneRM(for:
selectedExercise)))kg")
            .font(.system(size: 36, weight: .bold))
            .foregroundColor(AppColors.primary)
    }
    .frame(maxWidth: .infinity, alignment: .leading)
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

private var volumeCard: some View {
    VStack(alignment: .leading, spacing: 12) {
        Text("Total Volume (30 days)")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        Text("\(Int(dataManager.totalVolume(for: selectedExercise)))kg")
            .font(.system(size: 36, weight: .bold))
            .foregroundColor(AppColors.primary)
    }
    .frame(maxWidth: .infinity, alignment: .leading)
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

}

// MARK: - Profile View

struct ProfileView: View {
@EnvironmentObject var dataManager: DataManager

var body: some View {
    NavigationView {
        ScrollView {
            VStack(spacing: 20) {
                profileHeader
                settingsSection
            }
            .padding()
        }
        .navigationTitle("Profile")
        .background(AppColors.background.ignoresSafeArea())
    }
}

private var profileHeader: some View {
    VStack(spacing: 12) {
        Image(systemName: "person.circle.fill")
            .font(.system(size: 80))
            .foregroundColor(AppColors.primary)

        if let profile = dataManager.userProfile {
            Text(profile.name)
                .font(.system(size: 24, weight: .bold))
                .foregroundColor(AppColors.textPrimary)

            Text(profile.athleteLevel.rawValue)
                .font(.system(size: 16))
                .foregroundColor(AppColors.textSecondary)
        }
    }
    .frame(maxWidth: .infinity)
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

private var settingsSection: some View {
    VStack(spacing: 12) {
        NavigationLink(destination:
ProfileEditView().environmentObject(dataManager)) {
            SettingRow(icon: "person.fill", title: "Edit Profile", color:
.blue)
        }

        SettingRow(icon: "bell.fill", title: "Notifications", color:
```

```
.orange)
        SettingRow(icon: "chart.bar.fill", title: "Export Data", color:
.green)
        SettingRow(icon: "gear", title: "Settings", color: .gray)
    }
}

}

// MARK: - Helper Views

struct StatCard: View {
let title: String
let value: String
let icon: String
let color: Color

var body: some View {
    VStack(alignment: .leading, spacing: 8) {
        Image(systemName: icon)
            .font(.system(size: 24))
            .foregroundColor(color)

        Text(value)
            .font(.system(size: 20, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        Text(title)
            .font(.system(size: 12))
            .foregroundColor(AppColors.textSecondary)
    }
    .frame(maxWidth: .infinity, alignment: .leading)
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

}

struct WorkoutRowView: View {
let workout: Workout

var body: some View {
    HStack {
        Image(systemName: workout.exerciseType.icon)
            .foregroundColor(workout.exerciseType.color)
            .frame(width: 40, height: 40)
            .background(workout.exerciseType.color.opacity(0.2))
            .cornerRadius(8)

        VStack(alignment: .leading, spacing: 4) {
            Text(workout.exerciseType.rawValue)
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(AppColors.textPrimary)

            Text("\(workout.sets.count) sets • \
(Int(workout.totalVolume))kg volume")
                .font(.system(size: 14))
                .foregroundColor(AppColors.textSecondary)
        }

        Spacer()

        Text(workout.date, style: .date)
            .font(.system(size: 12))
            .foregroundColor(AppColors.textSecondary)
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

}

struct SettingRow: View {
let icon: String
let title: String
let color: Color

var body: some View {
    HStack {
        Image(systemName: icon)
            .foregroundColor(color)
            .frame(width: 40, height: 40)
            .background(color.opacity(0.2))
            .cornerRadius(8)
        Text(title)
            .font(.system(size: 16))
            .foregroundColor(AppColors.textPrimary)

        Spacer()

        Image(systemName: "chevron.right")
            .foregroundColor(AppColors.textSecondary)
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}
```

}

CONTINUA NEL PROSSIMO MESSAGGIO CON:

TodayWorkoutCard.swift

WorkoutPlannerView.swift

ExerciseDetailView.swift

ManualWorkoutCreatorView.swift

ProfileEditView.swift

Camera Views (tutti i file camera)

VideoAnalysisView.swift

VideoTrimmerView.swift

Vuoi che continui con gli altri file? 📂✨

Preparato utilizzando Claude Sonnet 4.5 Thinking

si

Perfetto! Continuo con tutti i file rimanenti! 🚀

📦 POWERLIFT APP - REMAINING FILES (PART 2)

FILE: TodayWorkoutCard.swift

PATH: Powerlift/Views/Dashboard/

```
swift
import SwiftUI

struct TodayWorkoutCard: View {
@EnvironmentObject var dataManager: DataManager
@State private var showWorkoutPlanner = false

var todayWorkout: Workout? {
    let today = Calendar.current.startOfDay(for: Date())
    return dataManager.workouts.first { workout in
        Calendar.current.isDate(workout.date, inSameDayAs: today)
    }
}

var body: some View {
    VStack(alignment: .leading, spacing: 16) {
        HStack {
            Text("Today's Workout")
                .font(.system(size: 22, weight: .bold))
                .foregroundColor(AppColors.textPrimary)

            Spacer()

            if todayWorkout == nil {
                Button(action: {
                    showWorkoutPlanner = true
                }) {
                    Image(systemName: "plus.circle.fill")
                        .font(.system(size: 24))
                        .foregroundColor(AppColors.primary)
                }
            }
        }

        if let workout = todayWorkout {
            completedWorkoutView(workout)
        } else {
            emptyWorkoutView
        }
    }
    .padding()
    .background(
        RoundedRectangle(cornerRadius: 16)
            .fill(AppColors.cardBackground)
    )
    .sheet(isPresented: $showWorkoutPlanner) {
        WorkoutPlannerView()
            .environmentObject(dataManager)
    }
}

// MARK: - Completed Workout View

private func completedWorkoutView(_ workout: Workout) -> some View {
    VStack(alignment: .leading, spacing: 12) {
        HStack {
            Image(systemName: workout.exerciseType.icon)
                .font(.system(size: 32))
                .foregroundColor(workout.exerciseType.color)

            VStack(alignment: .leading, spacing: 4) {
                Text(workout.exerciseType.rawValue)
                    .font(.system(size: 18, weight: .semibold))
                    .foregroundColor(AppColors.textPrimary)

                Text("\(workout.sets.count) sets completed")
                    .font(.system(size: 14))
                    .foregroundColor(AppColors.textSecondary)
            }

            Spacer()

            Image(systemName: "checkmark.circle.fill")
                .font(.system(size: 28))
                .foregroundColor(AppColors.success)
        }
        HStack(spacing: 20) {
            VStack(alignment: .leading, spacing: 4) {
                Text("Top Set")
                    .font(.system(size: 12))
                    .foregroundColor(AppColors.textSecondary)

                if let topSet = workout.topSet {
                    Text("\(Int(topSet.weight))kg × \(topSet.reps)")
                        .font(.system(size: 16, weight: .bold))
                        .foregroundColor(AppColors.textPrimary)
                }
            }

            VStack(alignment: .leading, spacing: 4) {
                Text("Total Volume")
                    .font(.system(size: 12))
                    .foregroundColor(AppColors.textSecondary)

                Text("\(Int(workout.totalVolume))kg")
                    .font(.system(size: 16, weight: .bold))
                    .foregroundColor(AppColors.textPrimary)
            }

            VStack(alignment: .leading, spacing: 4) {
                Text("Avg RPE")
                    .font(.system(size: 12))
                    .foregroundColor(AppColors.textSecondary)

                Text(String(format: "%.1f", workout.averageRPE))
                    .font(.system(size: 16, weight: .bold))
                    .foregroundColor(AppColors.rpeColor(for:
workout.averageRPE))
            }
        }
    }
}

// MARK: - Empty Workout View

private var emptyWorkoutView: some View {
    VStack(spacing: 16) {
        Image(systemName: "calendar.badge.clock")
            .font(.system(size: 48))
            .foregroundColor(AppColors.textSecondary)

        Text("No workout logged today")
            .font(.system(size: 16))
            .foregroundColor(AppColors.textSecondary)

        Button(action: {
            showWorkoutPlanner = true
        }) {
            Text("Start Workout")
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding()
                .background(AppColors.primary)
                .cornerRadius(12)
        }
    }
    .padding(.vertical, 20)
}
```

}

FILE: WorkoutPlannerView.swift

PATH: Powerlift/Views/Workout/

---

swift
import SwiftUI

struct WorkoutPlannerView: View {
@EnvironmentObject var dataManager: DataManager
@State private var selectedExercise: ExerciseType = .squat
@State private var showExerciseDetail = false
@State private var showManualCreator = false

```
var body: some View {
    NavigationView {
        ZStack {
            AppColors.background
                .ignoresSafeArea()

            ScrollView {
                VStack(spacing: 20) {
                    exerciseSelector

                    quickStatsCard

                    suggestedWorkoutCard

                    recentWorkoutsSection
                }
                .padding()
            }
        }
        .navigationTitle("Workout Planner")
        .navigationBarTitleDisplayMode(.large)
        .toolbar {
            ToolbarItem(placement: .navigationBarTrailing) {
                Button(action: {
                    showManualCreator = true
                }) {
                    Image(systemName: "plus")
                        .foregroundColor(AppColors.primary)
                }
            }
        }
        .sheet(isPresented: $showExerciseDetail) {
            ExerciseDetailView(exercise: selectedExercise)
                .environmentObject(dataManager)
        }
        .sheet(isPresented: $showManualCreator) {
            ManualWorkoutCreatorView(exercise: selectedExercise)
                .environmentObject(dataManager)
        }
    }
}
```

```
// MARK: - Exercise Selector

private var exerciseSelector: some View {
    VStack(alignment: .leading, spacing: 12) {
        Text("Select Exercise")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        HStack(spacing: 12) {
            ForEach(ExerciseType.allCases, id: \.self) { exercise in
                Button(action: {
                    withAnimation {
                        selectedExercise = exercise
                    }
                }) {
                    VStack(spacing: 8) {
                        Image(systemName: exercise.icon)
                            .font(.system(size: 32))
                            .foregroundColor(selectedExercise == exercise
? .white : exercise.color)

                        Text(exercise.rawValue)
                            .font(.system(size: 12, weight: .semibold))
                            .foregroundColor(selectedExercise == exercise
? .white : AppColors.textPrimary)
                    }
                    .frame(maxWidth: .infinity)
                    .padding()
                    .background(
                        RoundedRectangle(cornerRadius: 12)
                            .fill(selectedExercise == exercise ?
exercise.color : AppColors.cardBackground)
                    )
                }
            }
        }
    }
}

// MARK: - Quick Stats Card
```

```
private var quickStatsCard: some View {
    VStack(alignment: .leading, spacing: 12) {
        Text("Current Stats")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        HStack(spacing: 20) {
            VStack(alignment: .leading, spacing: 4) {
                Text("Estimated 1RM")
                    .font(.system(size: 12))
                    .foregroundColor(AppColors.textSecondary)

                Text("\(Int(dataManager.estimatedOneRM(for:
selectedExercise)))kg")
                    .font(.system(size: 20, weight: .bold))
                    .foregroundColor(AppColors.primary)
            }

            Divider()

            VStack(alignment: .leading, spacing: 4) {
                Text("Last Top Set")
                    .font(.system(size: 12))
                    .foregroundColor(AppColors.textSecondary)

                if let topSet = dataManager.topSet(for: selectedExercise)
{
                    Text("\(Int(topSet.weight))kg × \(topSet.reps)")
                        .font(.system(size: 20, weight: .bold))
                        .foregroundColor(AppColors.textPrimary)
                } else {
                    Text("No data")
                        .font(.system(size: 16))
                        .foregroundColor(AppColors.textSecondary)
                }
            }
        }
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

// MARK: - Suggested Workout Card

private var suggestedWorkoutCard: some View {
    VStack(alignment: .leading, spacing: 16) {
        HStack {
            Text("Suggested Workout")
                .font(.system(size: 18, weight: .semibold))
                .foregroundColor(AppColors.textPrimary)

            Spacer()

            Image(systemName: "sparkles")
                .foregroundColor(AppColors.primary)
        }

        if let profile = dataManager.userProfile {
            let volume = ProgressionAlgorithm.recommendVolume(
                exercise: selectedExercise,
                athleteLevel: profile.athleteLevel,
                trainingPhase: .strength
            )

            VStack(alignment: .leading, spacing: 8) {
                Text("Recommended: \(volume.sets) sets of \
(volume.reps.lowerBound)-\(volume.reps.upperBound) reps")
                    .font(.system(size: 14))
                    .foregroundColor(AppColors.textSecondary)

                Text("Target RPE: 7-8")
                    .font(.system(size: 14))
                    .foregroundColor(AppColors.textSecondary)
            }
        }

        Button(action: {
            showExerciseDetail = true
        }) {
            Text("Start Guided Workout")
```

```
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding()
                .background(AppColors.primary)
                .cornerRadius(12)
        }
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

// MARK: - Recent Workouts Section

private var recentWorkoutsSection: some View {
    VStack(alignment: .leading, spacing: 12) {
        Text("Recent \(selectedExercise.rawValue) Sessions")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        let workouts = dataManager.workouts(for:
selectedExercise).prefix(5)

        if workouts.isEmpty {
            Text("No previous workouts")
                .font(.system(size: 14))
                .foregroundColor(AppColors.textSecondary)
                .frame(maxWidth: .infinity)
                .padding(.vertical, 40)
        } else {
            ForEach(Array(workouts)) { workout in
                WorkoutHistoryRow(workout: workout)
            }
        }
    }
}

}

// MARK: - Workout History Row
```

```
struct WorkoutHistoryRow: View {
let workout: Workout

var body: some View {
    HStack {
        VStack(alignment: .leading, spacing: 4) {
            Text(workout.date, style: .date)
                .font(.system(size: 14, weight: .semibold))
                .foregroundColor(AppColors.textPrimary)

            Text("\(workout.sets.count) sets")
                .font(.system(size: 12))
                .foregroundColor(AppColors.textSecondary)
        }

        Spacer()

        if let topSet = workout.topSet {
            VStack(alignment: .trailing, spacing: 4) {
                Text("\(Int(topSet.weight))kg × \(topSet.reps)")
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textPrimary)

                Text("RPE \(String(format: "%.1f", topSet.rpe))")
                    .font(.system(size: 12))
                    .foregroundColor(AppColors.rpeColor(for: topSet.rpe))
            }
        }
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

}
```

FILE: ExerciseDetailView.swift

PATH: Powerlift/Views/Workout/

---

swift

```
import SwiftUI

struct ExerciseDetailView: View {
@EnvironmentObject var dataManager: DataManager
@Environment(.presentationMode) var presentationMode

let exercise: ExerciseType

@State private var sets: [WorkoutSet] = []
@State private var currentSetNumber = 1
@State private var weight = ""
@State private var reps = ""
@State private var rpe: Double = 7.0
@State private var isWarmup = false
@State private var notes = ""

var body: some View {
    NavigationView {
        ZStack {
            AppColors.background
                .ignoresSafeArea()

            ScrollView {
                VStack(spacing: 20) {
                    headerCard

                    inputSection

                    setsListSection
                }
                .padding()
            }
        }
        .navigationTitle(exercise.rawValue)
        .navigationBarTitleDisplayMode(.inline)
        .toolbar {
            ToolbarItem(placement: .navigationBarLeading) {
                Button("Cancel") {
                    presentationMode.wrappedValue.dismiss()
                }
                .foregroundColor(AppColors.textSecondary)
            }

            ToolbarItem(placement: .navigationBarTrailing) {
                Button("Finish") {
                    finishWorkout()
                }
                .foregroundColor(AppColors.primary)
                .fontWeight(.semibold)
                .disabled(sets.isEmpty)
            }
        }
    }
}

// MARK: - Header Card

private var headerCard: some View {
    VStack(spacing: 12) {
        Image(systemName: exercise.icon)
            .font(.system(size: 48))
            .foregroundColor(exercise.color)

        Text("Set \(currentSetNumber)")
            .font(.system(size: 24, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        if let lastWorkout = dataManager.workouts(for: exercise).first,
           let lastTopSet = lastWorkout.topSet {
            Text("Last: \(Int(lastTopSet.weight))kg × \(lastTopSet.reps) @
RPE \(String(format: "%.1f", lastTopSet.rpe))")
                .font(.system(size: 14))
                .foregroundColor(AppColors.textSecondary)
        }
    }
    .frame(maxWidth: .infinity)
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(16)
}

// MARK: - Input Section
```

```
private var inputSection: some View {
    VStack(spacing: 16) {
        HStack(spacing: 12) {
            VStack(alignment: .leading, spacing: 8) {
                Text("Weight (kg)")
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textSecondary)

                TextField("0", text: $weight)
                    .keyboardType(.decimalPad)
                    .font(.system(size: 20, weight: .semibold))
                    .foregroundColor(AppColors.textPrimary)
                    .padding()
                    .background(AppColors.cardBackgroundLight)
                    .cornerRadius(8)
            }

            VStack(alignment: .leading, spacing: 8) {
                Text("Reps")
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textSecondary)

                TextField("0", text: $reps)
                    .keyboardType(.numberPad)
                    .font(.system(size: 20, weight: .semibold))
                    .foregroundColor(AppColors.textPrimary)
                    .padding()
                    .background(AppColors.cardBackgroundLight)
                    .cornerRadius(8)
            }
        }

        VStack(alignment: .leading, spacing: 8) {
            HStack {
                Text("RPE")
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textSecondary)

                Spacer()
                Text(String(format: "%.1f", rpe))
                    .font(.system(size: 20, weight: .bold))
                    .foregroundColor(AppColors.rpeColor(for: rpe))
            }

            Slider(value: $rpe, in: 0...10, step: 0.5)
                .accentColor(AppColors.rpeColor(for: rpe))
        }
        .padding()
        .background(AppColors.cardBackground)
        .cornerRadius(12)

        Toggle("Warmup Set", isOn: $isWarmup)
            .font(.system(size: 16))
            .foregroundColor(AppColors.textPrimary)
            .padding()
            .background(AppColors.cardBackground)
            .cornerRadius(12)

        Button(action: addSet) {
            Text("Add Set")
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding()
                .background(AppColors.primary)
                .cornerRadius(12)
        }
        .disabled(weight.isEmpty || reps.isEmpty)
    }
}

// MARK: - Sets List Section

private var setsListSection: some View {
    VStack(alignment: .leading, spacing: 12) {
        if !sets.isEmpty {
            Text("Completed Sets")
                .font(.system(size: 18, weight: .semibold))
                .foregroundColor(AppColors.textPrimary)
            ForEach(Array(sets.enumerated()), id: \.element.id) { index,
set in
                SetRow(set: set, onDelete: {
                    sets.remove(at: index)
                    currentSetNumber = sets.count + 1
                })
            }
        }
    }
}

// MARK: - Actions

private func addSet() {
    guard let weightValue = Double(weight),
          let repsValue = Int(reps) else {
        return
    }

    let newSet = WorkoutSet(
        setNumber: currentSetNumber,
        weight: weightValue,
        reps: repsValue,
        rpe: rpe,
        isWarmup: isWarmup
    )

    sets.append(newSet)
    currentSetNumber += 1

    // Reset inputs
    weight = ""
    reps = ""
    isWarmup = false
}

private func finishWorkout() {
    let workout = Workout(
        date: Date(),
        exerciseType: exercise,
        sets: sets,
        notes: notes.isEmpty ? nil : notes
    )

    dataManager.addWorkout(workout)
    presentationMode.wrappedValue.dismiss()
}

}

// MARK: - Set Row

struct SetRow: View {
let set: WorkoutSet
let onDelete: () -> Void

var body: some View {
    HStack {
        VStack(alignment: .leading, spacing: 4) {
            HStack {
                Text("Set \(set.setNumber)")
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textPrimary)

                if set.isWarmup {
                    Text("WARMUP")
                        .font(.system(size: 10, weight: .bold))
                        .foregroundColor(.white)
                        .padding(.horizontal, 8)
                        .padding(.vertical, 2)
                        .background(AppColors.info)
                        .cornerRadius(4)
                }
            }

            Text("\(Int(set.weight))kg × \(set.reps) reps")
                .font(.system(size: 16, weight: .bold))
                .foregroundColor(AppColors.textPrimary)
        }

        Spacer()

        VStack(alignment: .trailing, spacing: 4) {
            Text("RPE \(String(format: "%.1f", set.rpe))")
```

```
                .font(.system(size: 14, weight: .semibold))
                .foregroundColor(AppColors.rpeColor(for: set.rpe))

            Text("Est. 1RM: \(Int(set.estimatedOneRM))kg")
                .font(.system(size: 12))
                .foregroundColor(AppColors.textSecondary)
        }

        Button(action: onDelete) {
            Image(systemName: "trash")
                .foregroundColor(.red)
                .padding(.leading, 8)
        }
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}
```

}

FILE: ManualWorkoutCreatorView.swift

PATH: Powerlift/Views/Workout/

---

swift
import SwiftUI

struct ManualWorkoutCreatorView: View {
@EnvironmentObject var dataManager: DataManager
@Environment(.presentationMode) var presentationMode

```
let exercise: ExerciseType

@State private var selectedDate = Date()
@State private var sets: [WorkoutSet] = []
@State private var notes = ""
@State private var showAddSetSheet = false
```

```
var body: some View {
    NavigationView {
        ZStack {
            AppColors.background
                .ignoresSafeArea()

            ScrollView {
                VStack(spacing: 20) {
                    dateSection

                    exerciseSection

                    setsSection

                    notesSection
                }
                .padding()
            }
        }
        .navigationTitle("Log Workout")
        .navigationBarTitleDisplayMode(.inline)
        .toolbar {
            ToolbarItem(placement: .navigationBarLeading) {
                Button("Cancel") {
                    presentationMode.wrappedValue.dismiss()
                }
                .foregroundColor(AppColors.textSecondary)
            }

            ToolbarItem(placement: .navigationBarTrailing) {
                Button("Save") {
                    saveWorkout()
                }
                .foregroundColor(AppColors.primary)
                .fontWeight(.semibold)
                .disabled(sets.isEmpty)
            }
        }
        .sheet(isPresented: $showAddSetSheet) {
            AddSetSheet(exercise: exercise, onAdd: { set in
                sets.append(set)
            })
        }
    }
}

// MARK: - Date Section

private var dateSection: some View {
    VStack(alignment: .leading, spacing: 8) {
        Text("Date")
            .font(.system(size: 14, weight: .semibold))
            .foregroundColor(AppColors.textSecondary)

        DatePicker("", selection: $selectedDate, displayedComponents:
.date)
            .datePickerStyle(GraphicalDatePickerStyle())
            .accentColor(AppColors.primary)
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

// MARK: - Exercise Section

private var exerciseSection: some View {
    HStack {
        Image(systemName: exercise.icon)
            .font(.system(size: 32))
            .foregroundColor(exercise.color)
            .frame(width: 60, height: 60)
            .background(exercise.color.opacity(0.2))
            .cornerRadius(12)

        VStack(alignment: .leading, spacing: 4) {
            Text(exercise.rawValue)
                .font(.system(size: 18, weight: .semibold))
                .foregroundColor(AppColors.textPrimary)

            Text("Main Lift")
                .font(.system(size: 14))
                .foregroundColor(AppColors.textSecondary)
        }

        Spacer()
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

// MARK: - Sets Section

private var setsSection: some View {
    VStack(alignment: .leading, spacing: 12) {
        HStack {
            Text("Sets")
                .font(.system(size: 18, weight: .semibold))
                .foregroundColor(AppColors.textPrimary)

            Spacer()

            Button(action: {
                showAddSetSheet = true
            }) {
                Image(systemName: "plus.circle.fill")
                    .font(.system(size: 24))
                    .foregroundColor(AppColors.primary)
            }
        }

        if sets.isEmpty {
            Text("No sets added yet")
                .font(.system(size: 14))
                .foregroundColor(AppColors.textSecondary)
                .frame(maxWidth: .infinity)
                .padding(.vertical, 40)
        } else {
            ForEach(Array(sets.enumerated()), id: \.element.id) { index,
set in
                SetRow(set: set, onDelete: {
                    sets.remove(at: index)
                })
            }
        }
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

// MARK: - Notes Section

private var notesSection: some View {
    VStack(alignment: .leading, spacing: 8) {
        Text("Notes (Optional)")
            .font(.system(size: 14, weight: .semibold))
            .foregroundColor(AppColors.textSecondary)

        TextEditor(text: $notes)
            .frame(height: 100)
            .padding(8)
            .background(AppColors.cardBackgroundLight)
            .cornerRadius(8)
            .foregroundColor(AppColors.textPrimary)
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

// MARK: - Save Workout

private func saveWorkout() {
    let workout = Workout(
        date: selectedDate,
        exerciseType: exercise,
        sets: sets,
        notes: notes.isEmpty ? nil : notes
    )

    dataManager.addWorkout(workout)
    presentationMode.wrappedValue.dismiss()
}

}

// MARK: - Add Set Sheet

struct AddSetSheet: View {
@Environment(.presentationMode) var presentationMode

let exercise: ExerciseType
let onAdd: (WorkoutSet) -> Void

@State private var weight = ""
@State private var reps = ""
@State private var rpe: Double = 7.0
@State private var isWarmup = false

var body: some View {
    NavigationView {
        ZStack {
            AppColors.background
                .ignoresSafeArea()

            VStack(spacing: 20) {
                HStack(spacing: 12) {
                    VStack(alignment: .leading, spacing: 8) {
                        Text("Weight (kg)")
                            .font(.system(size: 14, weight: .semibold))
                            .foregroundColor(AppColors.textSecondary)

                        TextField("0", text: $weight)
                            .keyboardType(.decimalPad)
                            .font(.system(size: 20, weight: .semibold))
                            .foregroundColor(AppColors.textPrimary)
                            .padding()
                            .background(AppColors.cardBackgroundLight)
                            .cornerRadius(8)
                    }

                    VStack(alignment: .leading, spacing: 8) {
                        Text("Reps")
                            .font(.system(size: 14, weight: .semibold))
                            .foregroundColor(AppColors.textSecondary)

                        TextField("0", text: $reps)
                            .keyboardType(.numberPad)
                            .font(.system(size: 20, weight: .semibold))
                            .foregroundColor(AppColors.textPrimary)
                            .padding()
                            .background(AppColors.cardBackgroundLight)
                            .cornerRadius(8)
                    }
                }

                VStack(alignment: .leading, spacing: 8) {
                    HStack {
                        Text("RPE")
                            .font(.system(size: 14, weight: .semibold))
                            .foregroundColor(AppColors.textSecondary)

                        Spacer()

                        Text(String(format: "%.1f", rpe))
                            .font(.system(size: 20, weight: .bold))
                            .foregroundColor(AppColors.rpeColor(for: rpe))
                    }

                    Slider(value: $rpe, in: 0...10, step: 0.5)
                        .accentColor(AppColors.rpeColor(for: rpe))
                }
                .padding()
                .background(AppColors.cardBackground)
                .cornerRadius(12)

                Toggle("Warmup Set", isOn: $isWarmup)
                    .font(.system(size: 16))
                    .foregroundColor(AppColors.textPrimary)
                    .padding()
                    .background(AppColors.cardBackground)
                    .cornerRadius(12)

                Spacer()
            }
```

```
            .padding()
        }
        .navigationTitle("Add Set")
        .navigationBarTitleDisplayMode(.inline)
        .toolbar {
            ToolbarItem(placement: .navigationBarLeading) {
                Button("Cancel") {
                    presentationMode.wrappedValue.dismiss()
                }
                .foregroundColor(AppColors.textSecondary)
            }

            ToolbarItem(placement: .navigationBarTrailing) {
                Button("Add") {
                    addSet()
                }
                .foregroundColor(AppColors.primary)
                .fontWeight(.semibold)
                .disabled(weight.isEmpty || reps.isEmpty)
            }
        }
    }
}

private func addSet() {
    guard let weightValue = Double(weight),
          let repsValue = Int(reps) else {
        return
    }

    let newSet = WorkoutSet(
        setNumber: 1,
        weight: weightValue,
        reps: repsValue,
        rpe: rpe,
        isWarmup: isWarmup
    )

    onAdd(newSet)
    presentationMode.wrappedValue.dismiss()
}
```

}

FILE: ProfileEditView.swift

PATH: Powerlift/Views/Profile/

---

swift
import SwiftUI

struct ProfileEditView: View {
@EnvironmentObject var dataManager: DataManager
@Environment(.presentationMode) var presentationMode

```
@State private var name = ""
@State private var age = ""
@State private var weight = ""
@State private var height = ""
@State private var selectedGender: UserProfile.Gender = .male
@State private var selectedLevel: AthleteLevel = .beginner
@State private var squat1RM = ""
@State private var bench1RM = ""
@State private var deadlift1RM = ""

var body: some View {
    ZStack {
        AppColors.background
            .ignoresSafeArea()

        ScrollView {
            VStack(spacing: 20) {
                profileImageSection

                personalInfoSection

                athleteLevelSection

                maxesSection
            }
            .padding()
        }
    }
    .navigationTitle("Edit Profile")
    .navigationBarTitleDisplayMode(.inline)
    .toolbar {
        ToolbarItem(placement: .navigationBarTrailing) {
            Button("Save") {
                saveProfile()
            }
            .foregroundColor(AppColors.primary)
            .fontWeight(.semibold)
        }
    }
    .onAppear(perform: loadProfile)
}

// MARK: - Profile Image Section

private var profileImageSection: some View {
    VStack(spacing: 12) {
        Image(systemName: "person.circle.fill")
            .font(.system(size: 100))
            .foregroundColor(AppColors.primary)

        Button("Change Photo") {
            // TODO: Implement photo picker
        }
        .font(.system(size: 14))
        .foregroundColor(AppColors.primary)
    }
    .frame(maxWidth: .infinity)
    .padding()
}

// MARK: - Personal Info Section

private var personalInfoSection: some View {
    VStack(alignment: .leading, spacing: 16) {
        Text("Personal Information")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)
```

```
VStack(alignment: .leading, spacing: 8) {
    Text("Name")
        .font(.system(size: 14, weight: .semibold))
        .foregroundColor(AppColors.textSecondary)

    TextField("Enter your name", text: $name)
        .textFieldStyle(ProfileTextFieldStyle())
}

HStack(spacing: 16) {
    VStack(alignment: .leading, spacing: 8) {
        Text("Age")
            .font(.system(size: 14, weight: .semibold))
            .foregroundColor(AppColors.textSecondary)

        TextField("Age", text: $age)
            .keyboardType(.numberPad)
            .textFieldStyle(ProfileTextFieldStyle())
    }

    VStack(alignment: .leading, spacing: 8) {
        Text("Weight (kg)")
            .font(.system(size: 14, weight: .semibold))
            .foregroundColor(AppColors.textSecondary)

        TextField("Weight", text: $weight)
            .keyboardType(.decimalPad)
            .textFieldStyle(ProfileTextFieldStyle())
    }
}

VStack(alignment: .leading, spacing: 8) {
    Text("Height (cm)")
        .font(.system(size: 14, weight: .semibold))
        .foregroundColor(AppColors.textSecondary)

    TextField("Height", text: $height)
        .keyboardType(.numberPad)
        .textFieldStyle(ProfileTextFieldStyle())
}
```

```
        VStack(alignment: .leading, spacing: 8) {
            Text("Gender")
                .font(.system(size: 14, weight: .semibold))
                .foregroundColor(AppColors.textSecondary)

            Picker("Gender", selection: $selectedGender) {
                Text("Male").tag(UserProfile.Gender.male)
                Text("Female").tag(UserProfile.Gender.female)
                Text("Other").tag(UserProfile.Gender.other)
            }
            .pickerStyle(SegmentedPickerStyle())
        }
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

// MARK: - Athlete Level Section

private var athleteLevelSection: some View {
    VStack(alignment: .leading, spacing: 16) {
        Text("Experience Level")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        Picker("Level", selection: $selectedLevel) {
            ForEach(AthleteLevel.allCases, id: \.self) { level in
                Text(level.rawValue).tag(level)
            }
        }
        .pickerStyle(SegmentedPickerStyle())

        Text(selectedLevel.description)
            .font(.system(size: 14))
            .foregroundColor(AppColors.textSecondary)
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
```

```
}

// MARK: - Maxes Section

private var maxesSection: some View {
    VStack(alignment: .leading, spacing: 16) {
        Text("Current 1RM (Optional)")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        VStack(spacing: 12) {
            HStack {
                Image(systemName: "figure.strengthtraining.traditional")
                    .foregroundColor(.blue)
                    .frame(width: 30)

                Text("Squat")
                    .foregroundColor(AppColors.textPrimary)

                Spacer()

                TextField("kg", text: $squat1RM)
                    .keyboardType(.decimalPad)
                    .multilineTextAlignment(.trailing)
                    .frame(width: 100)
                    .textFieldStyle(ProfileTextFieldStyle())
            }

            Divider()

            HStack {
                Image(systemName: "figure.cooldown")
                    .foregroundColor(.green)
                    .frame(width: 30)

                Text("Bench Press")
                    .foregroundColor(AppColors.textPrimary)

                Spacer()

                TextField("kg", text: $bench1RM)
```

```
                    .keyboardType(.decimalPad)
                    .multilineTextAlignment(.trailing)
                    .frame(width: 100)
                    .textFieldStyle(ProfileTextFieldStyle())
            }

            Divider()

            HStack {
                Image(systemName: "figure.stand")
                    .foregroundColor(.red)
                    .frame(width: 30)

                Text("Deadlift")
                    .foregroundColor(AppColors.textPrimary)

                Spacer()

                TextField("kg", text: $deadlift1RM)
                    .keyboardType(.decimalPad)
                    .multilineTextAlignment(.trailing)
                    .frame(width: 100)
                    .textFieldStyle(ProfileTextFieldStyle())
            }
        }
    }
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}

// MARK: - Load Profile

private func loadProfile() {
    guard let profile = dataManager.userProfile else { return }

    name = profile.name
    age = String(profile.age)
    weight = String(profile.weight)
    height = String(profile.heightCm)
    selectedGender = profile.gender
    selectedLevel = profile.athleteLevel

    if let squat = profile.squat1RM {
        squat1RM = String(squat)
    }
    if let bench = profile.bench1RM {
        bench1RM = String(bench)
    }
    if let deadlift = profile.deadlift1RM {
        deadlift1RM = String(deadlift)
    }
}

// MARK: - Save Profile

private func saveProfile() {
    let profile = UserProfile(
        name: name,
        age: Int(age) ?? 25,
        weight: Double(weight) ?? 75.0,
        heightCm: Double(height) ?? 175.0,
        gender: selectedGender,
        athleteLevel: selectedLevel,
        goals: dataManager.userProfile?.goals ?? [],
        squat1RM: Double(squat1RM),
        bench1RM: Double(bench1RM),
        deadlift1RM: Double(deadlift1RM)
    )

    dataManager.updateProfile(profile)
    presentationMode.wrappedValue.dismiss()
}

}

// MARK: - Profile Text Field Style

struct ProfileTextFieldStyle: TextFieldStyle {
func _body(configuration: TextField<Self._Label>) -> some View {
configuration
.padding()
.background(AppColors.cardBackgroundLight)
.cornerRadius(8)
```

```
.foregroundColor(AppColors.textPrimary)
}
}
```

CONTINUA con i file Camera nel prossimo messaggio! 📹

Vuoi che continui con CameraView, CameraRecordingView, VideoPlayerView, etc.? 🎥

Preparato utilizzando Claude Sonnet 4.5 Thinking
si dammeli tutti
Perfetto! Ecco tutti i file rimanenti della Camera e Video! 🎥

📦 POWERLIFT APP - CAMERA & VIDEO FILES (PART 3)

FILE: CameraView.swift

PATH: Powerlift/Views/Camera/

---

swift

```
import SwiftUI
import AVFoundation

struct CameraView: View {
@StateObject private var cameraManager = CameraManager()
@Environment(.presentationMode) var presentationMode

var body: some View {
    ZStack {
        CameraPreview(cameraManager: cameraManager)
            .ignoresSafeArea()

        VStack {
            HStack {
                Button(action: {
                    presentationMode.wrappedValue.dismiss()
                }) {
                    Image(systemName: "xmark")
                        .font(.system(size: 20))
                        .foregroundColor(.white)
                        .padding()
                        .background(Color.black.opacity(0.5))
                        .clipShape(Circle())
                }
                .padding()

                Spacer()
            }

            Spacer()

            Button(action: {
                cameraManager.capturePhoto()
            }) {
                Circle()
                    .fill(Color.white)
                    .frame(width: 70, height: 70)
                    .overlay(
                        Circle()
                            .stroke(Color.white, lineWidth: 3)
                            .frame(width: 80, height: 80)
                    )
            }
            .padding(.bottom, 40)
        }
    }
    .onAppear {
        cameraManager.checkAuthorization()
    }
    .onDisappear {
        cameraManager.stopSession()
    }
}

}

struct CameraPreview: UIViewRepresentable {
let cameraManager: CameraManager

func makeUIView(context: Context) -> UIView {
    let view = UIView(frame: .zero)
    view.backgroundColor = .black
    if let previewLayer = cameraManager.getPreviewLayer() {
        previewLayer.frame = view.bounds
        view.layer.addSublayer(previewLayer)
        context.coordinator.previewLayer = previewLayer
    }

    return view
}

func updateUIView(_ uiView: UIView, context: Context) {
    DispatchQueue.main.async {
        context.coordinator.previewLayer?.frame = uiView.bounds
    }
}

func makeCoordinator() -> Coordinator {
    Coordinator()
}

class Coordinator {
    var previewLayer: AVCaptureVideoPreviewLayer?
}
```

}

FILE: CameraRecordingView.swift

PATH: Powerlift/Views/Camera/

```
swift
import SwiftUI
import AVKit

struct CameraRecordingView: View {
@StateObject private var recordingManager = VideoRecordingManager()
@State private var showingVideoPicker = false
@State private var showingVideoPlayer = false
@Environment(.presentationMode) var presentationMode
```

```
var body: some View {
    ZStack {
        // Camera Preview
        if recordingManager.permissionGranted {
            CameraPreviewView(recordingManager: recordingManager)
                .ignoresSafeArea()
        } else {
            permissionView
        }

        // Controls Overlay
        VStack {
            topBar
            Spacer()
            bottomControls
        }
    }
    .onAppear {
        recordingManager.requestPermission()
    }
    .onDisappear {
        recordingManager.cleanup()
    }
    .sheet(isPresented: $showingVideoPicker) {
        VideoPickerView(onVideoPicked: handleVideoPicked)
    }
    .fullScreenCover(isPresented: $showingVideoPlayer) {
        if let videoURL = recordingManager.recordedVideoURL {
            VideoPlayerView(videoURL: videoURL, dataManager:
DataManager())
        }
    }
}

// MARK: - Handle Video Picked
private func handleVideoPicked(_ url: URL) {
    recordingManager.recordedVideoURL = url
    showingVideoPlayer = true
}

// MARK: - Permission View
```

```
private var permissionView: some View {
    ZStack {
        AppColors.background
            .ignoresSafeArea()

        VStack(spacing: 24) {
            Image(systemName: "camera.fill")
                .font(.system(size: 80))
                .foregroundColor(AppColors.textSecondary)

            Text("Accesso Camera Richiesto")
                .font(.system(size: 24, weight: .bold))
                .foregroundColor(AppColors.textPrimary)

            Text("Per registrare i tuoi allenamenti, abbiamo bisogno
dell'accesso alla camera")
                .font(.system(size: 16))
                .foregroundColor(AppColors.textSecondary)
                .multilineTextAlignment(.center)
                .padding(.horizontal, 40)

            Button(action: openSettings) {
                Text("Apri Impostazioni")
                    .font(.system(size: 16, weight: .semibold))
                    .foregroundColor(.white)
                    .frame(maxWidth: .infinity)
                    .padding()
                    .background(AppColors.primary)
                    .cornerRadius(12)
            }
            .padding(.horizontal, 40)
        }
    }
}

// MARK: - Top Bar
private var topBar: some View {
    HStack {
        Button(action: dismiss) {
            Image(systemName: "xmark")
                .font(.system(size: 20, weight: .semibold))
                .foregroundColor(.white)
                .frame(width: 44, height: 44)
                .background(Color.black.opacity(0.5))
                .clipShape(Circle())
        }

        Spacer()

        if recordingManager.isRecording {
            HStack(spacing: 8) {
                Circle()
                    .fill(Color.red)
                    .frame(width: 12, height: 12)

                Text("REC")
                    .font(.system(size: 16, weight: .bold))
                    .foregroundColor(.white)
            }
            .padding(.horizontal, 16)
            .padding(.vertical, 8)
            .background(Color.black.opacity(0.7))
            .cornerRadius(20)
        }

        Spacer()

        Button(action: {}) {
            Image(systemName: "grid")
                .font(.system(size: 20))
                .foregroundColor(.white)
                .frame(width: 44, height: 44)
                .background(Color.black.opacity(0.5))
                .clipShape(Circle())
        }
    }
    .padding()
}

// MARK: - Bottom Controls
private var bottomControls: some View {
    HStack(spacing: 40) {
        Button(action: { showingVideoPicker = true }) {
            Image(systemName: "photo.on.rectangle")
                .font(.system(size: 28))
                .foregroundColor(.white)
                .frame(width: 60, height: 60)
                .background(Color.black.opacity(0.5))
                .clipShape(Circle())
        }

        Button(action: toggleRecording) {
            ZStack {
                Circle()
                    .stroke(Color.white, lineWidth: 4)
                    .frame(width: 80, height: 80)

                if recordingManager.isRecording {
                    RoundedRectangle(cornerRadius: 8)
                        .fill(Color.red)
                        .frame(width: 40, height: 40)
                } else {
                    Circle()
                        .fill(Color.red)
                        .frame(width: 70, height: 70)
                }
            }
        }
        .disabled(!recordingManager.permissionGranted)

        Button(action: {}) {
            Image(systemName: "arrow.triangle.2.circlepath.camera")
                .font(.system(size: 28))
                .foregroundColor(.white)
                .frame(width: 60, height: 60)
                .background(Color.black.opacity(0.5))
                .clipShape(Circle())
        }
    }
    .padding(.bottom, 40)
}

// MARK: - Actions
```

```
private func dismiss() {
    presentationMode.wrappedValue.dismiss()
}

private func openSettings() {
    if let settingsUrl = URL(string: UIApplication.openSettingsURLString)
{
        UIApplication.shared.open(settingsUrl)
    }
}

private func toggleRecording() {
    if recordingManager.isRecording {
        recordingManager.stopRecording()
    } else {
        recordingManager.startRecording()
    }
}

}

// MARK: - Camera Preview
struct CameraPreviewView: UIViewRepresentable {
let recordingManager: VideoRecordingManager

func makeUIView(context: Context) -> UIView {
    let view = UIView(frame: .zero)
    view.backgroundColor = .black

    if let previewLayer = recordingManager.getPreviewLayer() {
        previewLayer.frame = view.bounds
        view.layer.addSublayer(previewLayer)
        context.coordinator.previewLayer = previewLayer
    }

    return view
}

func updateUIView(_ uiView: UIView, context: Context) {
    DispatchQueue.main.async {
        context.coordinator.previewLayer?.frame = uiView.bounds
    }
}
```

```
func makeCoordinator() -> Coordinator {
    Coordinator()
}

class Coordinator {
    var previewLayer: AVCaptureVideoPreviewLayer?
}
```

}

FILE: VideoPickerView.swift

PATH: Powerlift/Views/Camera/

---

swift
import SwiftUI
import PhotosUI

struct VideoPickerView: UIViewControllerRepresentable {
var selectedVideoURL: Binding<URL?>? = nil
var isLoading: Binding? = nil
var onVideoPicked: ((URL) -> Void)? = nil

```
@Environment(\.presentationMode) var presentationMode

// MARK: - Initializers

init(selectedVideoURL: Binding<URL?>, isLoading: Binding<Bool>) {
    self.selectedVideoURL = selectedVideoURL
    self.isLoading = isLoading
    self.onVideoPicked = nil
}

init(onVideoPicked: @escaping (URL) -> Void) {
    self.onVideoPicked = onVideoPicked
    self.selectedVideoURL = nil
    self.isLoading = nil
}
```

```
func makeUIViewController(context: Context) -> PHPickerViewController {
    var config = PHPickerConfiguration()
    config.filter = .videos
    config.selectionLimit = 1

    let picker = PHPickerViewController(configuration: config)
    picker.delegate = context.coordinator
    return picker
}

func updateUIViewController(_ uiViewController: PHPickerViewController,
context: Context) {}

func makeCoordinator() -> Coordinator {
    Coordinator(self)
}

class Coordinator: NSObject, PHPickerViewControllerDelegate {
    let parent: VideoPickerView

    init(_ parent: VideoPickerView) {
        self.parent = parent
    }

    func picker(_ picker: PHPickerViewController, didFinishPicking
results: [PHPickerResult]) {
        parent.presentationMode.wrappedValue.dismiss()

        guard let provider = results.first?.itemProvider,

provider.hasItemConformingToTypeIdentifier(UTType.movie.identifier) else {
            return
        }

        if let isLoading = parent.isLoading {
            DispatchQueue.main.async {
                isLoading.wrappedValue = true
            }
        }
        provider.loadFileRepresentation(forTypeIdentifier:
UTType.movie.identifier) { url, error in
            guard let url = url else {
                if let isLoading = self.parent.isLoading {
                    DispatchQueue.main.async {
                        isLoading.wrappedValue = false
                    }
                }
                return
            }

            let tempURL = FileManager.default.temporaryDirectory
                .appendingPathComponent(UUID().uuidString)
                .appendingPathExtension("mov")

            do {
                try FileManager.default.copyItem(at: url, to: tempURL)

                DispatchQueue.main.async {
                    if let selectedVideoURL = self.parent.selectedVideoURL
{
                        selectedVideoURL.wrappedValue = tempURL
                    }

                    if let isLoading = self.parent.isLoading {
                        isLoading.wrappedValue = false
                    }

                    self.parent.onVideoPicked?(tempURL)
                }
            } catch {
                print("Error copying video: \(error)")

                DispatchQueue.main.async {
                    if let isLoading = self.parent.isLoading {
                        isLoading.wrappedValue = false
                    }
                }
            }
        }
```

```
    }
}
```

}

FILE: VideoPlayerView.swift

PATH: Powerlift/Views/Camera/

---

swift

```
import SwiftUI
import AVKit

struct VideoPlayerView: View {
let videoURL: URL
@ObservedObject var dataManager: DataManager
@Environment(.presentationMode) var presentationMode
```

```
@State private var player: AVPlayer?
@State private var analysisResults: VideoAnalysis?
@State private var isAnalyzing = false

// MARK: - Init
init(videoURL: URL, dataManager: DataManager) {
    self.videoURL = videoURL
    self._dataManager = ObservedObject(wrappedValue: dataManager)
}

var body: some View {
    ZStack {
        AppColors.background
            .ignoresSafeArea()

        VStack(spacing: 0) {
            topBar

            if let player = player {
                VideoPlayer(player: player)
                    .frame(height: 400)
                    .cornerRadius(16)
                    .padding()
            }

            if let analysis = analysisResults {
                analysisView(analysis)
            } else if isAnalyzing {
                analyzingView
            } else {
                analyzeButton
            }

            Spacer()
        }
    }
    .onAppear {
        setupPlayer()
    }
}

// MARK: - Top Bar
private var topBar: some View {
    HStack {
        Button("Chiudi") {
            presentationMode.wrappedValue.dismiss()
        }
        .foregroundColor(AppColors.primary)

        Spacer()

        Text("Analisi Video")
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        Spacer()

        Button("Salva") {
            saveVideo()
        }
        .foregroundColor(AppColors.primary)
        .fontWeight(.semibold)
    }
    .padding()
}

// MARK: - Analyze Button
private var analyzeButton: some View {
    Button(action: analyzeVideo) {
        HStack {
            Image(systemName: "waveform.path.ecg")
            Text("Analizza Movimento")
        }
        .font(.system(size: 16, weight: .semibold))
        .foregroundColor(.white)
        .frame(maxWidth: .infinity)
        .padding()
        .background(AppColors.primary)
        .cornerRadius(12)
    }
    .padding()
}

// MARK: - Analyzing View
private var analyzingView: some View {
    VStack(spacing: 16) {
        ProgressView()
            .scaleEffect(1.5)

        Text("Analizzando il video...")
            .font(.system(size: 16))
            .foregroundColor(AppColors.textSecondary)
    }
    .frame(maxWidth: .infinity)
    .padding(.vertical, 40)
}

// MARK: - Analysis View
private func analysisView(_ analysis: VideoAnalysis) -> some View {
    VStack(alignment: .leading, spacing: 20) {
        Text("Risultati Analisi")
            .font(.system(size: 20, weight: .bold))
            .foregroundColor(AppColors.textPrimary)
        HStack(spacing: 12) {
            AnalysisCard(
                title: "Rep Count",
                value: "\(analysis.repCount)",
                icon: "repeat",
                color: .blue
            )

            AnalysisCard(
                title: "Velocità Media",
                value: String(format: "%.2f m/s",
analysis.averageVelocity),
                icon: "speedometer",
                color: .orange
            )
        }

        HStack(spacing: 12) {
            AnalysisCard(
                title: "ROM",
                value: "\(Int(analysis.rangeOfMotion)) cm",
                icon: "arrow.up.and.down",
                color: .green
            )

            AnalysisCard(
                title: "Durata",
                value: String(format: "%.1f s", analysis.duration),
                icon: "clock",
                color: .purple
            )
        }
    }
    .padding()
}

// MARK: - Setup Player
private func setupPlayer() {
    player = AVPlayer(url: videoURL)
    player?.play()
```

```
}

// MARK: - Analyze Video
private func analyzeVideo() {
    isAnalyzing = true

    // Simulate analysis (replace with actual video analysis)
    DispatchQueue.main.asyncAfter(deadline: .now() + 2) {
        analysisResults = VideoAnalysis(
            repCount: 5,
            averageVelocity: 0.75,
            rangeOfMotion: 45.5,
            duration: 12.3
        )
        isAnalyzing = false
    }
}

// MARK: - Save Video
private func saveVideo() {
    // TODO: Save video to workout
    presentationMode.wrappedValue.dismiss()
}

}

// MARK: - Video Analysis Model
struct VideoAnalysis {
let repCount: Int
let averageVelocity: Double
let rangeOfMotion: Double
let duration: Double
}

// MARK: - Analysis Card
struct AnalysisCard: View {
let title: String
let value: String
let icon: String
let color: Color

var body: some View {
    VStack(alignment: .leading, spacing: 8) {
        Image(systemName: icon)
            .font(.system(size: 24))
            .foregroundColor(color)

        Text(value)
            .font(.system(size: 20, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        Text(title)
            .font(.system(size: 12))
            .foregroundColor(AppColors.textSecondary)
    }
    .frame(maxWidth: .infinity, alignment: .leading)
    .padding()
    .background(AppColors.cardBackground)
    .cornerRadius(12)
}
```

}

FILE: VideoAnalysisView.swift

PATH: Powerlift/Views/Camera/

```
swift
import SwiftUI
import AVKit

struct VideoAnalysisView: View {
let videoURL: URL
@EnvironmentObject var dataManager: DataManager
@Environment(.presentationMode) var presentationMode
```

```
@State private var player: AVPlayer?
@State private var isAnalyzing = false
@State private var analysisComplete = false
@State private var repCount = 0
@State private var averageVelocity: Double = 0
@State private var peakVelocity: Double = 0
```

```
@State private var rangeOfMotion: Double = 0

var body: some View {
    ZStack {
        AppColors.background
            .ignoresSafeArea()

        VStack(spacing: 20) {
            // Video Player
            if let player = player {
                VideoPlayer(player: player)
                    .frame(height: 300)
                    .cornerRadius(16)
                    .padding()
            }

            // Analysis Section
            if isAnalyzing {
                analysingView
            } else if analysisComplete {
                resultsView
            } else {
                startAnalysisButton
            }

            Spacer()
        }
    }
    .navigationTitle("Video Analysis")
    .navigationBarTitleDisplayMode(.inline)
    .onAppear(perform: setupPlayer)
}

// MARK: - Analyzing View

private var analysingView: some View {
    VStack(spacing: 20) {
        ProgressView()
            .scaleEffect(1.5)

        Text("Analyzing movement...")
```

```
            .font(.system(size: 18, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        Text("This may take a few moments")
            .font(.system(size: 14))
            .foregroundColor(AppColors.textSecondary)
    }
    .frame(maxWidth: .infinity)
    .padding(.vertical, 60)
}

// MARK: - Results View

private var resultsView: some View {
    VStack(spacing: 20) {
        Text("Analysis Results")
            .font(.system(size: 24, weight: .bold))
            .foregroundColor(AppColors.textPrimary)

        VStack(spacing: 16) {
            ResultRow(title: "Repetitions", value: "\(repCount)", icon: "repeat", color: .blue)
            ResultRow(title: "Average Velocity", value: String(format: "%.2f m/s", averageVelocity), icon: "speedometer", color: .green)
            ResultRow(title: "Peak Velocity", value: String(format: "%.2f m/s", peakVelocity), icon: "bolt.fill", color: .orange)
            ResultRow(title: "Range of Motion", value: String(format: "%.1f cm", rangeOfMotion), icon: "arrow.up.and.down", color: .purple)
        }
        .padding()
        .background(AppColors.cardBackground)
        .cornerRadius(16)

        Button(action: saveAnalysis) {
            Text("Save to Workout")
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding()
                .background(AppColors.primary)
                .cornerRadius(12)
        }
        .padding(.horizontal)
    }
}

// MARK: - Start Analysis Button

private var startAnalysisButton: some View {
    VStack(spacing: 20) {
        Image(systemName: "waveform.path.ecg")
            .font(.system(size: 64))
            .foregroundColor(AppColors.primary)

        Text("Ready to analyze")
            .font(.system(size: 20, weight: .semibold))
            .foregroundColor(AppColors.textPrimary)

        Button(action: startAnalysis) {
            Text("Start Analysis")
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding()
                .background(AppColors.primary)
                .cornerRadius(12)
        }
        .padding(.horizontal)
    }
    .padding(.vertical, 40)
}

// MARK: - Setup Player

private func setupPlayer() {
    player = AVPlayer(url: videoURL)
}

// MARK: - Start Analysis

private func startAnalysis() {
    isAnalyzing = true
    // Simulate analysis (in production, use BarbellTracker)
    DispatchQueue.main.asyncAfter(deadline: .now() + 3.0) {
        repCount = Int.random(in: 3...8)
        averageVelocity = Double.random(in: 0.5...1.2)
        peakVelocity = averageVelocity * 1.3
        rangeOfMotion = Double.random(in: 30...60)

        isAnalyzing = false
        analysisComplete = true
    }
}

// MARK: - Save Analysis

private func saveAnalysis() {
    // TODO: Save analysis results to workout
    presentationMode.wrappedValue.dismiss()
}

}

// MARK: - Result Row

struct ResultRow: View {
let title: String
let value: String
let icon: String
let color: Color

var body: some View {
    HStack {
        Image(systemName: icon)
            .font(.system(size: 24))
            .foregroundColor(color)
            .frame(width: 40)

        VStack(alignment: .leading, spacing: 4) {
            Text(title)
                .font(.system(size: 14))
                .foregroundColor(AppColors.textSecondary)

            Text(value)
```

```
                .font(.system(size: 20, weight: .bold))
                .foregroundColor(AppColors.textPrimary)
        }

        Spacer()
    }
    .padding()
    .background(AppColors.cardBackgroundLight)
    .cornerRadius(12)
}
```

}

FILE: VideoTrimmerView.swift

PATH: Powerlift/Views/Camera/

---

swift
import SwiftUI
import AVKit
import AVFoundation

struct VideoTrimmerView: View {
let videoURL: URL
@Environment(.presentationMode) var presentationMode

```
@State private var player: AVPlayer?
@State private var startTime: Double = 0
@State private var endTime: Double = 10
@State private var duration: Double = 10
@State private var isTrimming = false

var body: some View {
    NavigationView {
        ZStack {
            AppColors.background
                .ignoresSafeArea()

            VStack(spacing: 20) {
                // Video Player
                if let player = player {
                    VideoPlayer(player: player)
                        .frame(height: 300)
                        .cornerRadius(16)
                        .padding()
                }

                // Trimmer Controls
                trimmerControls

                // Time Display
                timeDisplay

                Spacer()

                // Action Buttons
                actionButtons
            }
            .padding()
        }
        .navigationTitle("Trim Video")
        .navigationBarTitleDisplayMode(.inline)
        .toolbar {
            ToolbarItem(placement: .navigationBarLeading) {
                Button("Cancel") {
                    presentationMode.wrappedValue.dismiss()
                }
                .foregroundColor(AppColors.textSecondary)
            }
        }
        .onAppear(perform: setupPlayer)
    }
}

// MARK: - Trimmer Controls

private var trimmerControls: some View {
    VStack(alignment: .leading, spacing: 12) {
        Text("Start Time")
            .font(.system(size: 14, weight: .semibold))
```

```
                .foregroundColor(AppColors.textSecondary)

            HStack {
                Slider(value: $startTime, in: 0...duration) { _ in
                    seekToTime(startTime)
                }
                .accentColor(AppColors.primary)

                Text(formatTime(startTime))
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textPrimary)
                    .frame(width: 60)
            }

            Text("End Time")
                .font(.system(size: 14, weight: .semibold))
                .foregroundColor(AppColors.textSecondary)
                .padding(.top, 8)

            HStack {
                Slider(value: $endTime, in: startTime...duration) { _ in
                    seekToTime(endTime)
                }
                .accentColor(AppColors.primary)

                Text(formatTime(endTime))
                    .font(.system(size: 14, weight: .semibold))
                    .foregroundColor(AppColors.textPrimary)
                    .frame(width: 60)
            }
        }
        .padding()
        .background(AppColors.cardBackground)
        .cornerRadius(12)
    }

    // MARK: - Time Display

    private var timeDisplay: some View {
        HStack(spacing: 40) {
            VStack(spacing: 4) {
                Text("Duration")
                    .font(.system(size: 12))
                    .foregroundColor(AppColors.textSecondary)

                Text(formatTime(endTime - startTime))
                    .font(.system(size: 20, weight: .bold))
                    .foregroundColor(AppColors.primary)
            }

            VStack(spacing: 4) {
                Text("Total")
                    .font(.system(size: 12))
                    .foregroundColor(AppColors.textSecondary)

                Text(formatTime(duration))
                    .font(.system(size: 20, weight: .bold))
                    .foregroundColor(AppColors.textPrimary)
            }
        }
        .frame(maxWidth: .infinity)
        .padding()
        .background(AppColors.cardBackground)
        .cornerRadius(12)
    }

    // MARK: - Action Buttons

    private var actionButtons: some View {
        VStack(spacing: 12) {
            Button(action: previewTrim) {
                HStack {
                    Image(systemName: "play.fill")
                    Text("Preview Trim")
                }
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(.white)
                .frame(maxWidth: .infinity)
                .padding()
                .background(AppColors.info)
                .cornerRadius(12)
            }
            Button(action: trimVideo) {
                if isTrimming {
                    ProgressView()
                        .progressViewStyle(CircularProgressViewStyle(tint:
    .white))
                } else {
                    HStack {
                        Image(systemName: "scissors")
                        Text("Trim & Save")
                    }
                }
            }
            .font(.system(size: 16, weight: .semibold))
            .foregroundColor(.white)
            .frame(maxWidth: .infinity)
            .padding()
            .background(AppColors.primary)
            .cornerRadius(12)
            .disabled(isTrimming)
        }
    }

    // MARK: - Setup Player

    private func setupPlayer() {
        player = AVPlayer(url: videoURL)

        if let asset = player?.currentItem?.asset {
            duration = CMTimeGetSeconds(asset.duration)
            endTime = duration
        }
    }

    // MARK: - Seek To Time

    private func seekToTime(_ time: Double) {
        let cmTime = CMTime(seconds: time, preferredTimescale: 600)
        player?.seek(to: cmTime)
    }
```

```
// MARK: - Preview Trim

private func previewTrim() {
    let startCMTime = CMTime(seconds: startTime, preferredTimescale: 600)
    player?.seek(to: startCMTime)
    player?.play()

    DispatchQueue.main.asyncAfter(deadline: .now() + (endTime -
startTime)) {
        player?.pause()
    }
}

// MARK: - Trim Video

private func trimVideo() {
    isTrimming = true

    guard let asset = player?.currentItem?.asset else { return }

    let startCMTime = CMTime(seconds: startTime, preferredTimescale: 600)
    let endCMTime = CMTime(seconds: endTime, preferredTimescale: 600)
    let timeRange = CMTimeRange(start: startCMTime, end: endCMTime)

    let outputURL = FileManager.default.temporaryDirectory
        .appendingPathComponent(UUID().uuidString)
        .appendingPathExtension("mov")

    guard let exportSession = AVAssetExportSession(asset: asset,
presetName: AVAssetExportPresetHighestQuality) else {
        isTrimming = false
        return
    }

    exportSession.outputURL = outputURL
    exportSession.outputFileType = .mov
    exportSession.timeRange = timeRange

    exportSession.exportAsynchronously {
        DispatchQueue.main.async {
            isTrimming = false
            switch exportSession.status {
            case .completed:
                // TODO: Save trimmed video
                presentationMode.wrappedValue.dismiss()
            case .failed, .cancelled:
                print("Export failed: \(String(describing:
exportSession.error))")
            default:
                break
            }
        }
    }
}

// MARK: - Format Time

private func formatTime(_ time: Double) -> String {
    let minutes = Int(time) / 60
    let seconds = Int(time) % 60
    let milliseconds = Int((time.truncatingRemainder(dividingBy: 1)) *
100)
    return String(format: "%02d:%02d.%02d", minutes, seconds,
milliseconds)
}
```

}

FILE: VideoPicker.swift (LEGACY - Optional)

PATH: Powerlift/Views/Camera/

---

swift
import SwiftUI
import PhotosUI

// Legacy VideoPicker - use VideoPickerView instead
// Kept for backwards compatibility

```
struct VideoPicker: UIViewControllerRepresentable {
@Binding var selectedVideoURL: URL?
@Binding var isLoading: Bool
@Environment(.presentationMode) var presentationMode

func makeUIViewController(context: Context) -> PHPickerViewController {
    var config = PHPickerConfiguration()
    config.filter = .videos
    config.selectionLimit = 1

    let picker = PHPickerViewController(configuration: config)
    picker.delegate = context.coordinator
    return picker
}

func updateUIViewController(_ uiViewController: PHPickerViewController,
context: Context) {}

func makeCoordinator() -> Coordinator {
    Coordinator(self)
}

class Coordinator: NSObject, PHPickerViewControllerDelegate {
    let parent: VideoPicker

    init(_ parent: VideoPicker) {
        self.parent = parent
    }

    func picker(_ picker: PHPickerViewController, didFinishPicking
results: [PHPickerResult]) {
        parent.presentationMode.wrappedValue.dismiss()

        guard let provider = results.first?.itemProvider,

provider.hasItemConformingToTypeIdentifier(UTType.movie.identifier) else {
            return
        }

        parent.isLoading = true

        provider.loadFileRepresentation(forTypeIdentifier:
```

```
UTType.movie.identifier) { url, error in
            guard let url = url else {
                DispatchQueue.main.async {
                    self.parent.isLoading = false
                }
                return
            }

            let tempURL = FileManager.default.temporaryDirectory
                .appendingPathComponent(UUID().uuidString)
                .appendingPathExtension("mov")

            do {
                try FileManager.default.copyItem(at: url, to: tempURL)

                DispatchQueue.main.async {
                    self.parent.selectedVideoURL = tempURL
                    self.parent.isLoading = false
                }
            } catch {
                print("Error copying video: \(error)")
                DispatchQueue.main.async {
                    self.parent.isLoading = false
                }
            }
        }
    }
}

}
```