AI Fitness Coach - Technical Specification for Development

Project Overview

Build an MVP mobile fitness app with real-time AI vision coaching that analyzes form, counts reps, and provides voice feedback.

Tech Stack

Mobile Framework

- **React Native** (recommended for MVP speed)
- Alternative: Flutter

AI/ML Vision

- **Google MediaPipe Pose Detection**
  - Library: `@mediapipe/pose` or `react-native-mediapipe`
  - 33 body landmarks in 3D coordinates
  - 30-60 FPS on-device processing

Backend (Minimal for MVP)

- **Firebase**
  - Authentication: Firebase Auth
  - Database: Firestore
  - Storage: For user profiles only (no video storage)
  - Cloud Functions: Optional for leaderboard aggregation

Additional Libraries

- Text-to-Speech: `react-native-tts` or `expo-speech`
- Local Storage: AsyncStorage / MMKV
- Camera: `react-native-vision-camera` or `expo-camera`
- State Management: React Context API (sufficient for MVP)

Core Features Implementation

1. Camera & Pose Detection Module

File: `src/services/PoseDetection.js`

javascript

```javascript
import { Pose } from '@mediapipe/pose';

class PoseDetectionService {
  constructor() {
    this.pose = new Pose({
      locateFile: (file) => `https://cdn.jsdelivr.net/npm/@mediapipe/pose/${file}`
    });

    this.pose.setOptions({
      modelComplexity: 1,
      smoothLandmarks: true,
      enableSegmentation: false,
      minDetectionConfidence: 0.5,
      minTrackingConfidence: 0.5
    });
  }

  async detectPose(imageData) {
    await this.pose.send({ image: imageData });
    return this.pose.results;
  }

  getLandmarks(results) {
    if (!results.poseLandmarks) return null;

    return {
      leftShoulder: results.poseLandmarks[11],
      rightShoulder: results.poseLandmarks[12],
      leftElbow: results.poseLandmarks[13],
      rightElbow: results.poseLandmarks[14],
      leftWrist: results.poseLandmarks[15],
      rightWrist: results.poseLandmarks[16],
      leftHip: results.poseLandmarks[23],
      rightHip: results.poseLandmarks[24],
      leftKnee: results.poseLandmarks[25],
      rightKnee: results.poseLandmarks[26],
      leftAnkle: results.poseLandmarks[27],
      rightAnkle: results.poseLandmarks[28]
    };
  }
}

export default new PoseDetectionService();
```

2. Form Analysis Engine

File: `src/services/FormAnalyzer.js`

javascript

```
class FormAnalyzer {
  // Calculate angle between three points
  calculateAngle(pointA, pointB, pointC) {
    const radians = Math.atan2(pointC.y - pointB.y, pointC.x - pointB.x) -
                    Math.atan2(pointA.y - pointB.y, pointA.x - pointB.x);
    let angle = Math.abs(radians * 180.0 / Math.PI);
    if (angle > 180.0) angle = 360 - angle;
    return angle;
  }

  // Analyze squat form
  analyzeSquat(landmarks) {
    const hipAngle = this.calculateAngle(
      landmarks.rightShoulder,
      landmarks.rightHip,
      landmarks.rightKnee
    );

    const kneeAngle = this.calculateAngle(
      landmarks.rightHip,
      landmarks.rightKnee,
      landmarks.rightAnkle
    );

    const errors = [];
    let formScore = 100;

    // Check depth
    if (hipAngle > 100) {
      errors.push({
        type: 'depth',
        message: 'Go deeper - hips below knees',
        severity: 'medium'
      });
      formScore -= 20;
    }

    // Check knee alignment
    const kneeDiff = Math.abs(landmarks.rightKnee.x - landmarks.rightAnkle.x);
    if (kneeDiff > 0.1) {
      errors.push({
        type: 'knee_alignment',
        message: 'Keep knees over toes',
        severity: 'high'
      });
```

```
    });
    formScore -= 30;
  }

  return {
    formScore,
    errors,
    isGoodForm: errors.length === 0
  };
}

// Analyze pushup form
analyzePushup(landmarks) {
  const elbowAngle = this.calculateAngle(
    landmarks.rightShoulder,
    landmarks.rightElbow,
    landmarks.rightWrist
  );

  // Check body alignment (plank position)
  const bodyAlignment = Math.abs(
    landmarks.rightShoulder.y - landmarks.rightHip.y
  );

  const errors = [];
  let formScore = 100;

  if (elbowAngle > 100) {
    errors.push({
      type: 'elbow_depth',
      message: 'Lower your chest more',
      severity: 'medium'
    });
    formScore -= 20;
  }

  if (bodyAlignment > 0.15) {
    errors.push({
      type: 'body_straight',
      message: 'Keep your body straight',
      severity: 'high'
    });
    formScore -= 30;
  }

  return { formScore, errors, isGoodForm: errors.length === 0 };
}
```

```
}

export default new FormAnalyzer();
```

---

3. Rep Counter State Machine

File: `src/services/RepCounter.js`

javascript

```
class RepCounter {
  constructor() {
    this.reset();
  }

  reset() {
    this.count = 0;
    this.state = 'standing'; // standing, descending, bottom, ascending
    this.landmarkHistory = [];
  }

  // Smooth landmarks over last N frames
  smoothLandmark(newValue, landmarkKey) {
    if (!this.landmarkHistory[landmarkKey]) {
      this.landmarkHistory[landmarkKey] = [];
    }

    this.landmarkHistory[landmarkKey].push(newValue);
    if (this.landmarkHistory[landmarkKey].length > 5) {
      this.landmarkHistory[landmarkKey].shift();
    }

    return this.landmarkHistory[landmarkKey].reduce((sum, val) => sum + val, 0)
           / this.landmarkHistory[landmarkKey].length;
  }

  updateSquat(landmarks, formAnalysis) {
    const hipY = this.smoothLandmark(landmarks.rightHip.y, 'hipY');
    const isLowPosition = hipY > 0.55;

    switch(this.state) {
      case 'standing':
        if (isLowPosition) this.state = 'descending';
        break;

      case 'descending':
        if (isLowPosition && this.isStable(hipY, 'hipY')) {
          this.state = 'bottom';
        }
        break;

      case 'bottom':
        if (!isLowPosition) this.state = 'ascending';
        break;
```

```
      case 'ascending':
        if (!isLowPosition && this.isStable(hipY, 'hipY')) {
          if (formAnalysis.isGoodForm) {
            this.count++;
            this.state = 'standing';
            return { repCompleted: true, isGoodRep: true };
          } else {
            this.state = 'standing';
            return { repCompleted: true, isGoodRep: false };
          }
        }
        break;
    }

    return { repCompleted: false };
  }

  isStable(currentValue, key) {
    const history = this.landmarkHistory[key] || [];
    if (history.length < 3) return false;

    const variance = history.reduce((sum, val) =>
      sum + Math.pow(val - currentValue, 2), 0
    ) / history.length;

    return variance < 0.001; // Adjust threshold as needed
  }

  getCount() {
    return this.count;
  }
}

export default RepCounter;
```

---

4. Voice Coaching System

File: `src/services/VoiceCoach.js`

javascript

```javascript
import Tts from 'react-native-tts';

class VoiceCoach {
  constructor() {
    this.personality = 'supportive'; // supportive, neutral, drill_sergeant
    this.lastCueTime = {};
    this.cooldownMs = 3000; // 3 seconds between same cue type

    this.cues = {
      supportive: {
        rep_complete: ['Nice!', 'Great form!', 'Keep it up!', 'Excellent!'],
        form_error: ['Small adjustment needed', 'Check your form'],
        motivation: ["You've got this!", 'Almost there!', 'Keep going!'],
        milestone: ['Halfway there!', "You're crushing it!"]
      },
      neutral: {
        rep_complete: ['Rep complete', 'Good', 'Continue'],
        form_error: ['Adjust form', 'Correction needed'],
        motivation: ['Continue', 'Maintain pace'],
        milestone: ['50% complete', '10 reps remaining']
      },
      drill_sergeant: {
        rep_complete: ["That's ONE!", 'AGAIN!', 'MOVE!'],
        form_error: ['FIX IT NOW!', 'FOCUS!'],
        motivation: ['PUSH!', "Don't quit!", 'FASTER!'],
        milestone: ["HALFWAY! DON'T STOP!", 'GIVE ME MORE!']
      }
    };
  }

  setPersonality(personality) {
    this.personality = personality;
  }

  speak(cueType) {
    const now = Date.now();
    if (this.lastCueTime[cueType] &&
        (now - this.lastCueTime[cueType]) < this.cooldownMs) {
      return; // Cooldown active
    }

    const messages = this.cues[this.personality][cueType];
    const message = messages[Math.floor(Math.random() * messages.length)];
```

```
    const rate = this.personality === 'drill_sergeant' ? 1.2 : 1.0;
    const pitch = this.personality === 'drill_sergeant' ? 0.9 : 1.0;

    Tts.speak(message, { rate, pitch });
    this.lastCueTime[cueType] = now;
  }

  onRepComplete(isGoodRep) {
    this.speak(isGoodRep ? 'rep_complete' : 'form_error');
  }

  onMilestone(currentReps, targetReps) {
    if (currentReps === Math.floor(targetReps / 2)) {
      this.speak('milestone');
    } else if (currentReps === targetReps - 3) {
      this.speak('motivation');
    }
  }
}

export default new VoiceCoach();
```

---

5. Main Workout Screen Component

File: `src/screens/LiveWorkoutScreen.js`

javascript

```javascript
import React, { useState, useEffect, useRef } from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { Camera } from 'react-native-vision-camera';
import PoseDetection from '../services/PoseDetection';
import FormAnalyzer from '../services/FormAnalyzer';
import RepCounter from '../services/RepCounter';
import VoiceCoach from '../services/VoiceCoach';

const LiveWorkoutScreen = ({ route, navigation }) => {
  const { exerciseType } = route.params; // 'squat', 'pushup', etc.
  const [repCount, setRepCount] = useState(0);
  const [formScore, setFormScore] = useState(100);
  const [feedback, setFeedback] = useState('');
  const [duration, setDuration] = useState(0);

  const cameraRef = useRef(null);
  const repCounterRef = useRef(new RepCounter());
  const startTimeRef = useRef(Date.now());

  useEffect(() => {
    const timer = setInterval(() => {
      setDuration(Math.floor((Date.now() - startTimeRef.current) / 1000));
    }, 1000);

    return () => clearInterval(timer);
  }, []);

  const processFrame = async (frame) => {
    // Get pose landmarks
    const results = await PoseDetection.detectPose(frame);
    const landmarks = PoseDetection.getLandmarks(results);

    if (!landmarks) return;

    // Analyze form based on exercise type
    let formAnalysis;
    if (exerciseType === 'squat') {
      formAnalysis = FormAnalyzer.analyzeSquat(landmarks);
    } else if (exerciseType === 'pushup') {
      formAnalysis = FormAnalyzer.analyzePushup(landmarks);
    }

    setFormScore(formAnalysis.formScore);

    // Update feedback
```

```
  // Update feedback
  if (formAnalysis.errors.length > 0) {
    setFeedback(formAnalysis.errors[0].message);
  } else {
    setFeedback('Perfect form!');
  }

  // Count reps
  const repUpdate = repCounterRef.current.updateSquat(landmarks, formAnalysis);

  if (repUpdate.repCompleted) {
    const newCount = repCounterRef.current.getCount();
    setRepCount(newCount);
    VoiceCoach.onRepComplete(repUpdate.isGoodRep);
    VoiceCoach.onMilestone(newCount, 20); // Assuming 20 rep target
  }

  // Draw overlay (skeleton)
  drawSkeletonOverlay(landmarks, formAnalysis);
};

const drawSkeletonOverlay = (landmarks, formAnalysis) => {
  // Implementation would use Canvas or SVG overlay
  // Pseudo-code for drawing logic
};

const handleEndWorkout = () => {
  const workoutData = {
    exerciseType,
    reps: repCount,
    duration,
    formScore,
    timestamp: new Date().toISOString()
  };

  navigation.navigate('WorkoutSummary', { workoutData });
};

return (
  <View style={styles.container}>
    <Camera
      ref={cameraRef}
      style={StyleSheet.absoluteFill}
      device={'front'}
      isActive={true}
      frameProcessor={processFrame}
    />
```

```
      {/* Overlay UI */}
      <View style={styles.overlay}>
        <View style={styles.topBar}>
          <Text style={styles.repCount}>{repCount}</Text>
          <Text style={styles.timer}>{formatTime(duration)}</Text>
        </View>

        <View style={styles.feedbackContainer}>
          <Text style={styles.feedback}>{feedback}</Text>
        </View>

        <View style={styles.formScoreContainer}>
          <Text style={styles.formScoreLabel}>FORM</Text>
          <Text style={styles.formScoreValue}>{formScore}%</Text>
        </View>

        <View style={styles.bottomBar}>
          <TouchableOpacity onPress={handleEndWorkout}>
            <Text style={styles.endButton}>End Workout</Text>
          </TouchableOpacity>
        </View>
      </View>
    </View>
  );
};

const formatTime = (seconds) => {
  const mins = Math.floor(seconds / 60);
  const secs = seconds % 60;
  return `${mins}:${secs.toString().padStart(2, '0')}`;
};

const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: 'black' },
  overlay: { flex: 1, justifyContent: 'space-between' },
  topBar: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    padding: 20,
    backgroundColor: 'rgba(0,0,0,0.6)'
  },
  repCount: { fontSize: 64, fontWeight: 'bold', color: 'white' },
  timer: { fontSize: 24, color: 'white' },
  feedbackContainer: {
    alignItems: 'center',
```

```
    paddingHorizontal: 20
  },
  feedback: {
    backgroundColor: '#10b981',
    color: 'white',
    paddingHorizontal: 24,
    paddingVertical: 12,
    borderRadius: 24,
    fontSize: 16,
    fontWeight: '600'
  },
  formScoreContainer: {
    position: 'absolute',
    left: 20,
    top: '50%',
    backgroundColor: 'rgba(0,0,0,0.6)',
    padding: 16,
    borderRadius: 16
  },
  formScoreLabel: { color: 'white', fontSize: 12 },
  formScoreValue: { color: '#10b981', fontSize: 32, fontWeight: 'bold' },
  bottomBar: {
    padding: 20,
    alignItems: 'center',
    backgroundColor: 'rgba(0,0,0,0.6)'
  },
  endButton: {
    backgroundColor: '#ef4444',
    color: 'white',
    paddingHorizontal: 32,
    paddingVertical: 12,
    borderRadius: 24,
    fontSize: 16,
    fontWeight: '600'
  }
});

export default LiveWorkoutScreen;
```

---

6. Data Models & Firebase Structure

Firestore Collections:

javascript

```
// users/{userId}
{
  id: "user123",
  name: "John Doe",
  email: "john@example.com",
  coachPersonality: "supportive", // supportive, neutral, drill_sergeant
  goal: "build_muscle", // build_muscle, lose_weight, stay_active, get_stronger
  createdAt: timestamp,
  currentStreak: 3,
  totalWorkouts: 47
}

// workouts/{workoutId}
{
  id: "workout123",
  userId: "user123",
  exerciseType: "squat",
  reps: 25,
  duration: 323, // seconds
  formScore: 92,
  calories: 47,
  timestamp: timestamp,
  date: "2025-09-30"
}

// leaderboard_weekly/{userId}
{
  userId: "user123",
  userName: "John Doe",
  totalReps: 247,
  weekStartDate: "2025-09-24",
  lastUpdated: timestamp
}
```

File: **src/services/FirebaseService.js**

javascript

```javascript
import firestore from '@react-native-firebase/firestore';
import auth from '@react-native-firebase/auth';

class FirebaseService {
  async saveWorkout(workoutData) {
    const userId = auth().currentUser.uid;

    await firestore()
      .collection('workouts')
      .add({
        ...workoutData,
        userId,
        timestamp: firestore.FieldValue.serverTimestamp()
      });

    // Update weekly leaderboard
    await this.updateLeaderboard(userId, workoutData.reps);
  }

  async updateLeaderboard(userId, reps) {
    const weekStart = this.getWeekStart();
    const leaderboardRef = firestore()
      .collection('leaderboard_weekly')
      .doc(userId);

    await leaderboardRef.set({
      userId,
      userName: auth().currentUser.displayName,
      totalReps: firestore.FieldValue.increment(reps),
      weekStartDate: weekStart,
      lastUpdated: firestore.FieldValue.serverTimestamp()
    }, { merge: true });
  }

  async getLeaderboard() {
    const weekStart = this.getWeekStart();

    const snapshot = await firestore()
      .collection('leaderboard_weekly')
      .where('weekStartDate', '==', weekStart)
      .orderBy('totalReps', 'desc')
      .limit(100)
      .get();
```

```
  return snapshot.docs.map(doc => doc.data());
}

async getUserWorkoutHistory(limit = 30) {
  const userId = auth().currentUser.uid;

  const snapshot = await firestore()
    .collection('workouts')
    .where('userId', '==', userId)
    .orderBy('timestamp', 'desc')
    .limit(limit)
    .get();

  return snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
}

async updateUserStreak() {
  const userId = auth().currentUser.uid;
  const userRef = firestore().collection('users').doc(userId);

  const doc = await userRef.get();
  const userData = doc.data();

  const today = new Date().toDateString();
  const lastWorkout = userData.lastWorkoutDate;

  let newStreak = userData.currentStreak || 0;

  if (lastWorkout) {
    const lastDate = new Date(lastWorkout).toDateString();
    const yesterday = new Date(Date.now() - 86400000).toDateString();

    if (lastDate === yesterday) {
      newStreak += 1;
    } else if (lastDate !== today) {
      newStreak = 1;
    }
  } else {
    newStreak = 1;
  }

  await userRef.update({
    currentStreak: newStreak,
    lastWorkoutDate: today,
    totalWorkouts: firestore.FieldValue.increment(1)
  });
}
```

```
  getWeekStart() {
    const now = new Date();
    const dayOfWeek = now.getDay();
    const diff = now.getDate() - dayOfWeek + (dayOfWeek === 0 ? -6 : 1);
    const monday = new Date(now.setDate(diff));
    monday.setHours(0, 0, 0, 0);
    return monday.toISOString().split('T')[0];
  }
}

export default new FirebaseService();
```

---

7. Exercise Configuration System

File: `src/config/exercises.json`

json


```
{
  "squat": {
    "name": "Squats",
    "difficulty": "Beginner",
    "targetMuscles": "Legs, Glutes",
    "durationEstimate": "5-10 min",
    "caloriesPerRep": 0.5,
    "landmarks": ["shoulder", "hip", "knee", "ankle"],
    "rules": [
      {
        "id": "depth",
        "type": "angle",
        "points": ["shoulder", "hip", "knee"],
        "min": 80,
        "max": 100,
        "errorMessage": "Squat deeper - hips below knees",
        "severity": "medium"
      },
      {
        "id": "knee_alignment",
        "type": "horizontal_distance",
        "points": ["knee", "ankle"],
        "maxDeviation": 0.1,
        "errorMessage": "Keep knees aligned with toes",
        "severity": "high"
      },
      {
        "id": "back_straight",
        "type": "angle",
        "points": ["shoulder", "hip", "knee"],
        "min": 160,
        "errorMessage": "Keep your chest up",
        "severity": "medium"
      }
    ],
    "repDetection": {
      "landmark": "hip",
      "axis": "y",
      "threshold": 0.55,
      "direction": "down_up"
    }
  },
  "pushup": {
    "name": "Push-ups",
```

```
    "difficulty": "Beginner",
    "targetMuscles": "Chest, Arms",
    "durationEstimate": "5-8 min",
    "caloriesPerRep": 0.4,
    "landmarks": ["shoulder", "elbow", "wrist", "hip", "ankle"],
    "rules": [
      {
        "id": "elbow_depth",
        "type": "angle",
        "points": ["shoulder", "elbow", "wrist"],
        "min": 70,
        "max": 90,
        "errorMessage": "Lower your chest more",
        "severity": "medium"
      },
      {
        "id": "body_straight",
        "type": "vertical_alignment",
        "points": ["shoulder", "hip", "ankle"],
        "maxDeviation": 0.15,
        "errorMessage": "Keep your body straight",
        "severity": "high"
      }
    ],
    "repDetection": {
      "landmark": "shoulder",
      "axis": "y",
      "threshold": 0.4,
      "direction": "down_up"
    }
  }
}
```

---

8. Local State Management (Context API)

File: `src/context/AppContext.js`

javascript

```javascript
import React, { createContext, useState, useContext, useEffect } from 'react';
import AsyncStorage from '@react-native-async-storage/async-storage';

const AppContext = createContext();

export const AppProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [coachPersonality, setCoachPersonality] = useState('supportive');
  const [workoutHistory, setWorkoutHistory] = useState([]);
  const [currentStreak, setCurrentStreak] = useState(0);

  useEffect(() => {
    loadUserPreferences();
  }, []);

  const loadUserPreferences = async () => {
    try {
      const personality = await AsyncStorage.getItem('coachPersonality');
      if (personality) setCoachPersonality(personality);
    } catch (error) {
      console.error('Error loading preferences:', error);
    }
  };

  const saveCoachPersonality = async (personality) => {
    setCoachPersonality(personality);
    await AsyncStorage.setItem('coachPersonality', personality);
  };

  const addWorkout = (workout) => {
    setWorkoutHistory(prev => [workout, ...prev]);
  };

  return (
    <AppContext.Provider value={{
      user,
      setUser,
      coachPersonality,
      saveCoachPersonality,
      workoutHistory,
      addWorkout,
      currentStreak,
      setCurrentStreak
    }}>
```

```
      {children}
    </AppContext.Provider>
  );
};

export const useApp = () => useContext(AppContext);
```

---

9. Navigation Structure

File: `src/navigation/AppNavigator.js`

javascript

```javascript
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

// Screens
import WelcomeScreen from '../screens/WelcomeScreen';
import GoalSelectionScreen from '../screens/GoalSelectionScreen';
import CoachPersonalityScreen from '../screens/CoachPersonalityScreen';
import CameraSetupScreen from '../screens/CameraSetupScreen';
import DashboardScreen from '../screens/DashboardScreen';
import WorkoutSelectionScreen from '../screens/WorkoutSelectionScreen';
import LiveWorkoutScreen from '../screens/LiveWorkoutScreen';
import WorkoutSummaryScreen from '../screens/WorkoutSummaryScreen';
import HistoryScreen from '../screens/HistoryScreen';
import LeaderboardScreen from '../screens/LeaderboardScreen';
import ProfileScreen from '../screens/ProfileScreen';

const Stack = createStackNavigator();
const Tab = createBottomTabNavigator();

const OnboardingStack = () => (
  <Stack.Navigator screenOptions={{ headerShown: false }}>
    <Stack.Screen name="Welcome" component={WelcomeScreen} />
    <Stack.Screen name="GoalSelection" component={GoalSelectionScreen} />
    <Stack.Screen name="CoachPersonality" component={CoachPersonalityScreen} />
    <Stack.Screen name="CameraSetup" component={CameraSetupScreen} />
  </Stack.Navigator>
);

const MainTabs = () => (
  <Tab.Navigator>
    <Tab.Screen name="Home" component={DashboardScreen} />
    <Tab.Screen name="History" component={HistoryScreen} />
    <Tab.Screen name="Leaderboard" component={LeaderboardScreen} />
    <Tab.Screen name="Profile" component={ProfileScreen} />
  </Tab.Navigator>
);

const AppNavigator = () => {
  const [isOnboarded, setIsOnboarded] = useState(false);

  return (
    <NavigationContainer>
```

```
      <Stack.Navigator screenOptions={{ headerShown: false }}>
        {!isOnboarded ? (
          <Stack.Screen name="Onboarding" component={OnboardingStack} />
        ) : (
          <>
            <Stack.Screen name="MainTabs" component={MainTabs} />
            <Stack.Screen name="WorkoutSelection" component={WorkoutSelectionScreen} />
            <Stack.Screen name="LiveWorkout" component={LiveWorkoutScreen} />
            <Stack.Screen name="WorkoutSummary" component={WorkoutSummaryScreen} />
          </>
        )}
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default AppNavigator;
```

---

10. Push Notifications

File: `src/services/NotificationService.js`

javascript

```javascript
import PushNotification from 'react-native-push-notification';
import PushNotificationIOS from '@react-native-community/push-notification-ios';

class NotificationService {
  configure() {
    PushNotification.configure({
      onNotification: function (notification) {
        notification.finish(PushNotificationIOS.FetchResult.NoData);
      },
      permissions: {
        alert: true,
        badge: true,
        sound: true,
      },
      popInitialNotification: true,
      requestPermissions: true,
    });

    this.createChannels();
  }

  createChannels() {
    PushNotification.createChannel({
      channelId: 'workout-reminders',
      channelName: 'Workout Reminders',
    });
  }

  scheduleDailyReminder(hour, minute) {
    PushNotification.localNotificationSchedule({
      channelId: 'workout-reminders',
      title: 'Time to train! 💪',
      message: 'Your body is ready for today\'s workout',
      date: this.getNextScheduleTime(hour, minute),
      repeatType: 'day',
      allowWhileIdle: true,
    });
  }

  scheduleStreakReminder(streakDays) {
    // Send if no workout in 24 hours
    PushNotification.localNotificationSchedule({
      channelId: 'workout-reminders',
      title: `Don't break your ${streakDays}-day streak!`,
```

```
      message: 'Quick 10-min session?',
      date: new Date(Date.now() + 24 * 60 * 60 * 1000),
      allowWhileIdle: true,
    });
  }

  getNextScheduleTime(hour, minute) {
    const now = new Date();
    const scheduledTime = new Date();
    scheduledTime.setHours(hour, minute, 0, 0);

    if (scheduledTime <= now) {
      scheduledTime.setDate(scheduledTime.getDate() + 1);
    }

    return scheduledTime;
  }

  cancelAllNotifications() {
    PushNotification.cancelAllLocalNotifications();
  }
}

export default new NotificationService();
```

---

11. Performance Optimization Tips

javascript

```javascript
// Frame processing optimization
const FRAME_SKIP = 2; // Process every 2nd frame
let frameCounter = 0;

const processFrame = async (frame) => {
  frameCounter++;
  if (frameCounter % FRAME_SKIP !== 0) return;

  // Process frame...
};

// Landmark smoothing to reduce jitter
class LandmarkSmoother {
  constructor(windowSize = 5) {
    this.windowSize = windowSize;
    this.history = {};
  }

  smooth(landmarks) {
    const smoothed = {};

    for (const [key, value] of Object.entries(landmarks)) {
      if (!this.history[key]) this.history[key] = [];

      this.history[key].push(value);
      if (this.history[key].length > this.windowSize) {
        this.history[key].shift();
      }

      smoothed[key] = this.average(this.history[key]);
    }

    return smoothed;
  }

  average(points) {
    const sum = points.reduce((acc, p) => ({
      x: acc.x + p.x,
      y: acc.y + p.y,
      z: acc.z + (p.z || 0)
    }), { x: 0, y: 0, z: 0 });

    return {
      x: sum.x / points.length,
```

```
      y: sum.y / points.length,
      z: sum.z / points.length
    };
  }
}
```

---

12. Testing Strategy

Unit Tests

- Form analysis angle calculations
- Rep counter state machine transitions
- Landmark smoothing algorithms

Integration Tests

- Complete workout flow
- Firebase data sync
- Leaderboard updates

Performance Tests

- Frame processing latency (target: <33ms)
- Memory usage during 10-min workout
- Battery consumption

---

13. Deployment Checklist

Pre-Launch

- [ ] Camera permissions configured (iOS Info.plist, Android manifest)
- [ ] Firebase project setup (auth, firestore rules)
- [ ] MediaPipe models bundled or CDN configured
- [ ] TTS permissions and initialization
- [ ] App icons and splash screens
- [ ] Privacy policy URL (for camera usage)

iOS Specific

- [ ] Info.plist camera usage description
- [ ] Background modes if needed
- [ ] TestFlight beta testing

Android Specific

- [ ] Camera and microphone permissions in manifest
- [ ] ProGuard rules for MediaPipe
- [ ] Google Play internal testing

---

14. Future Enhancements (Post-MVP)

1. **Multi-sport expansion** (tennis, yoga, boxing)
2. **Workout programs** (4-week plans)
3. **Social features** (friend challenges, sharing)
4. **Wearable integration** (Apple Watch, Fitbit)
5. **Advanced analytics** (progress charts, muscle group balance)
6. **Custom AI model training** (per-user form optimization)
7. **Offline mode** (all features work without internet)
8. **Premium subscription** (advanced exercises, meal plans)

Quick Start Commands

```bash
# Initialize React Native project
npx react-native init AIFitnessCoach
cd AIFitnessCoach

# Install dependencies
npm install @mediapipe/pose
npm install @react-native-firebase/app @react-native-firebase/firestore
npm install @react-native-firebase/auth
npm install react-native-tts
npm install react-native-vision-camera
npm install @react-navigation/native @react-navigation/stack
npm install @react-navigation/bottom-tabs
npm install @react-native-async-storage/async-storage
npm install react-native-push-notification

# iOS specific
cd ios && pod install && cd ..

# Run
npm run ios
# or
npm run android
```

Key Metrics to Track

1. **User Retention**: 3+ sessions in first week (target: >40%)
2. **Workout Completion Rate**: % of started workouts finished (target: >75%)
3. **Form Accuracy**: Average form score (target: >85%)
4. **Session Duration**: Average workout length (target: 8-12 min)
5. **Streak Maintenance**: Users with 3+ day streaks (target: >30%)

Support & Resources

- MediaPipe Docs: https://developers.google.com/mediapipe
- React Native: https://reactnative.dev
- Firebase: https://firebase.google.com/docs
- Vision Camera: https://github.com/mrousavy/react-native-vision-camera

---

END OF TECHNICAL SPECIFICATION