# PraxisForma Style Guide

## 1. Introduction

This style guide establishes comprehensive coding standards for PraxisForma's AI-powered coaching platform. All developers and AI coding assistants must strictly adhere to these guidelines to ensure consistent, maintainable, secure, and high-quality code across our mobile applications, backend services, and AI/ML components.

## 2. General Principles

### 2.1 Core Values

- **Youth Safety First**: Every code decision must prioritize the safety and privacy of young athletes
- **Privacy by Design**: Implement privacy protection at the code level, not as an afterthought
- **Coach Amplification**: Build tools that enhance human coaching, never replace it
- **Performance Obsession**: Mobile-first performance optimization in every component
- **Accessibility**: Ensure all features work for athletes with varying abilities
- **International Ready**: Write code that supports global deployment from day one

### 2.2 Code Quality Standards

- **Zero Technical Debt**: Address code quality issues immediately, never accumulate debt
- **Test-First Development**: Write tests before implementation for all new features
- **Documentation Required**: Every function, class, and API must have comprehensive documentation
- **Security by Default**: Implement secure patterns and never compromise on security for convenience

## 3. TypeScript/JavaScript Standards

### 3.1 Code Formatting

**Formatter**: Prettier with the following configuration:

json

```
{
  "semi": true,
  "trailingComma": "es5",
  "singleQuote": true,
  "printWidth": 100,
  "tabWidth": 2,
  "useTabs": false
}
```

**Linting**: ESLint with strict TypeScript rules:

json

```
{
  "extends": [
    "@typescript-eslint/recommended",
    "@typescript-eslint/recommended-requiring-type-checking",
    "prettier"
  ],
  "rules": {
    "@typescript-eslint/no-unused-vars": "error",
    "@typescript-eslint/explicit-function-return-type": "error",
    "@typescript-eslint/no-explicit-any": "error"
  }
}
```

## 3.2 Naming Conventions

**Variables and Functions**: camelCase

typescript

```
// ✅ DO
const athleteScore = calculatePowerQuotient(movement);
const processVideoAnalysis = async (videoFile: File): Promise<AnalysisResult> => {};

// ❌ DON'T
const athlete_score = calculate_power_quotient(movement);
const ProcessVideoAnalysis = async (videoFile: File) => {};
```

**Constants**: SCREAMING_SNAKE_CASE

```
typescript

// ✅ DO
const MAX_VIDEO_DURATION_SECONDS = 60;
const SUPPORTED_VIDEO_FORMATS = ['mp4', 'mov', 'avi'];

// ❌ DON'T
const maxVideoDurationSeconds = 60;
const supportedVideoFormats = ['mp4', 'mov', 'avi'];
```

**Types and Interfaces**: PascalCase with descriptive names

```
typescript

// ✅ DO
interface AthleteProfile {
  id: string;
  name: string;
  sport: SportType;
  birthDate: Date;
}

type PowerQuotientScore = {
  overall: number;
  technique: number;
  power: number;
  consistency: number;
};

// ❌ DON'T
interface athleteProfile {
  id: string;
  name: string;
}

type PQS = {
  score: number;
};
```

**Enums**: PascalCase with descriptive values

typescript

```typescript
// ✅ DO
enum SportType {
  ShotPut = 'SHOT_PUT',
  Discus = 'DISCUS',
  StrengthTraining = 'STRENGTH_TRAINING',
}

enum AnalysisStatus {
  Pending = 'PENDING',
  Processing = 'PROCESSING',
  Completed = 'COMPLETED',
  Failed = 'FAILED',
}

// ❌ DON'T
enum sport {
  shotput = 'shotput',
  discus = 'discus',
}
```

## 3.3 File and Folder Structure

**File Naming**: kebab-case for files, PascalCase for components

```
// ✅ DO
src/
├── components/
│   ├── AthleteProfile.tsx
│   ├── VideoAnalysis.tsx
│   └── CoachDashboard.tsx
├── services/
│   ├── video-processing.service.ts
│   ├── athlete-analytics.service.ts
│   └── coaching-ai.service.ts
├── utils/
│   ├── biomechanics-calculator.ts
│   ├── privacy-protection.ts
│   └── form-validation.ts

// ❌ DON'T
src/
├── Components/
│   ├── athleteProfile.tsx
│   └── videoAnalysis.tsx
├── Services/
│   ├── videoProcessing.service.ts
│   └── athleteAnalytics.service.ts
```

**Folder Organization**:

```
src/
├── components/          # Reusable UI components
│   ├── common/          # Shared components across sports
│   ├── sport-specific/  # Sport-specific UI components
│   └── coach/           # Coach-specific dashboard components
├── services/            # Business logic and API services
│   ├── ai/              # AI/ML related services
│   ├── auth/            # Authentication services
│   └── data/            # Data access layer
├── types/               # TypeScript type definitions
│   ├── api/             # API response types
│   ├── biomechanics/    # Biomechanical analysis types
│   └── user/            # User and athlete types
├── utils/               # Pure utility functions
├── constants/           # Application constants
├── hooks/               # Custom React hooks
└── screens/             # Mobile app screens
```

### 3.4 TypeScript Usage

**Strict Type Safety**: Always use explicit types, never `any`

```typescript
// ✅ DO
interface VideoAnalysisRequest {
  videoUrl: string;
  sportType: SportType;
  athleteId: string;
  analysisOptions: AnalysisOptions;
}

const analyzeVideo = async (
  request: VideoAnalysisRequest
): Promise<AnalysisResult> => {
  // Implementation
};

// ❌ DON'T
const analyzeVideo = async (request: any): Promise<any> => {
  // Implementation
};
```

### Union Types for Sport-Specific Data:

typescript

```
// ✅ DO
type ThrowingAnalysis = {
  type: 'THROWING';
  releaseAngle: number;
  powerTransfer: number;
  footwork: FootworkAnalysis;
};

type LiftingAnalysis = {
  type: 'LIFTING';
  formScore: number;
  safetyWarnings: string[];
  muscleActivation: MuscleActivationData;
};

type BiomechanicalAnalysis = ThrowingAnalysis | LiftingAnalysis;

// ❌ DON'T
type Analysis = {
  type: string;
  data: any;
};
```

### Generic Types for Reusable Components:

```typescript
// ✅ DO
interface ApiResponse<T> {
  success: boolean;
  data: T;
  error?: string;
  timestamp: Date;
}

interface SportSpecificScore<T extends SportType> {
  sportType: T;
  score: T extends SportType.ShotPut ? PowerQuotientScore : LiftQuotientScore;
}

// ❌ DON'T
interface ApiResponse {
  success: boolean;
  data: any;
  error?: string;
}
```

# 4. React Native & Component Standards

## 4.1 Component Structure

**Functional Components with Hooks**: Use only functional components

typescript

```typescript
// ✅ DO
interface AthleteProfileProps {
  athleteId: string;
  onScoreUpdate: (score: number) => void;
}

const AthleteProfile: React.FC<AthleteProfileProps> = ({ athleteId, onScoreUpdate }) => {
  const [athlete, setAthlete] = useState<AthleteData | null>(null);
  const [loading, setLoading] = useState<boolean>(true);

  useEffect(() => {
    loadAthleteData(athleteId);
  }, [athleteId]);

  const loadAthleteData = async (id: string): Promise<void> => {
    // Implementation
  };

  return (
    <View style={styles.container}>
      {/* Component JSX */}
    </View>
  );
};

// ❌ DON'T
class AthleteProfile extends React.Component {
  // Class components are not allowed
}
```

**Component Props**: Always define explicit prop interfaces

typescript

```
// ✅ DO
interface VideoPlayerProps {
  videoUrl: string;
  analysisOverlay?: BiomechanicalOverlay;
  onPlaybackComplete: () => void;
  showControls?: boolean;
  autoPlay?: boolean;
}

// ❌ DON'T
const VideoPlayer = (props: any) => {
  // Implementation
};
```

## 4.2 State Management

**Local State**: Use `useState` for component-specific state

typescript

```
// ✅ DO
const [analysisResult, setAnalysisResult] = useState<AnalysisResult | null>(null);
const [isProcessing, setIsProcessing] = useState<boolean>(false);
const [errorMessage, setErrorMessage] = useState<string>('');

// ❌ DON'T
const [state, setState] = useState<any>({});
```

**Global State**: Use Redux Toolkit for app-wide state

typescript

```typescript
// ✅ DO - Redux Slice
import { createSlice, PayloadAction } from '@reduxjs/toolkit';

interface AthleteState {
  currentAthlete: AthleteProfile | null;
  recentAnalyses: AnalysisResult[];
  loading: boolean;
  error: string | null;
}

const athleteSlice = createSlice({
  name: 'athlete',
  initialState,
  reducers: {
    setCurrentAthlete: (state, action: PayloadAction<AthleteProfile>) => {
      state.currentAthlete = action.payload;
      state.error = null;
    },
    addAnalysisResult: (state, action: PayloadAction<AnalysisResult>) => {
      state.recentAnalyses.unshift(action.payload);
      if (state.recentAnalyses.length > 10) {
        state.recentAnalyses.pop();
      }
    },
  },
});

// ❌ DON'T
const athleteReducer = (state: any, action: any) => {
  // Untyped reducer
};
```

## 4.3 Styling Standards

**StyleSheet Usage**: Use React Native StyleSheet for all styling

typescript

```typescript
// ✅ DO
import { StyleSheet, ViewStyle, TextStyle } from 'react-native';

interface Styles {
  container: ViewStyle;
  title: TextStyle;
  scoreDisplay: ViewStyle;
  errorText: TextStyle;
}

const styles = StyleSheet.create<Styles>({
  container: {
    flex: 1,
    backgroundColor: '#FFFFFF',
    padding: 16,
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    color: '#333333',
    marginBottom: 16,
  },
  scoreDisplay: {
    backgroundColor: '#F0F8FF',
    borderRadius: 8,
    padding: 12,
    marginVertical: 8,
  },
  errorText: {
    color: '#FF6B6B',
    fontSize: 14,
    textAlign: 'center',
  },
});

// ❌ DON'T
const styles = {
  container: {
    flex: 1,
  },
  // Inline styles without types
};
```

**Design Tokens**: Use consistent design tokens

typescript

```typescript
// ✅ DO - Design Tokens
export const Colors = {
  primary: '#007AFF',
  secondary: '#34C759',
  error: '#FF3B30',
  warning: '#FF9500',
  background: '#F8F9FA',
  surface: '#FFFFFF',
  text: {
    primary: '#000000',
    secondary: '#8E8E93',
    inverse: '#FFFFFF',
  },
} as const;

export const Spacing = {
  xs: 4,
  sm: 8,
  md: 16,
  lg: 24,
  xl: 32,
} as const;

export const Typography = {
  h1: { fontSize: 32, fontWeight: 'bold' as const },
  h2: { fontSize: 24, fontWeight: 'bold' as const },
  body: { fontSize: 16, fontWeight: 'normal' as const },
  caption: { fontSize: 14, fontWeight: 'normal' as const },
} as const;

// ❌ DON'T
const styles = StyleSheet.create({
  text: {
    color: '#333', // Magic numbers
    fontSize: 16,
  },
});
```

## 5. Backend API Standards

## 5.1 RESTful API Design

**URL Structure**: Use consistent, descriptive endpoints

```typescript
// ✅ DO
GET    /api/v1/athletes/{athleteId}/analyses
POST   /api/v1/athletes/{athleteId}/analyses
GET    /api/v1/athletes/{athleteId}/analyses/{analysisId}
PUT    /api/v1/athletes/{athleteId}/analyses/{analysisId}
DELETE /api/v1/athletes/{athleteId}/analyses/{analysisId}

GET    /api/v1/coaches/{coachId}/teams
POST   /api/v1/coaches/{coachId}/teams/{teamId}/athletes

// ❌ DON'T
GET /api/getAthleteAnalyses?id=123
POST /api/createAnalysis
GET /analysis/123/get
```

**Response Format**: Consistent API response structure

```typescript
// ✅ DO
interface ApiResponse<T> {
  success: boolean;
  data: T;
  message?: string;
  error?: {
    code: string;
    message: string;
    details?: Record<string, unknown>;
  };
  meta?: {
    timestamp: string;
    requestId: string;
    version: string;
  };
}

interface PaginatedResponse<T> extends ApiResponse<T[]> {
  pagination: {
    page: number;
    limit: number;
    total: number;
    hasNext: boolean;
    hasPrevious: boolean;
  };
}

// ❌ DON'T
interface ApiResponse {
  status: string;
  result: any;
  msg?: string;
}
```

## 5.2 Input Validation

**Request Validation**: Use comprehensive validation schemas

typescript

```typescript
// ✅ DO
import Joi from 'joi';

const createAnalysisSchema = Joi.object({
  athleteId: Joi.string().uuid().required(),
  videoUrl: Joi.string().uri().required(),
  sportType: Joi.string().valid('SHOT_PUT', 'DISCUS', 'STRENGTH_TRAINING').required(),
  analysisOptions: Joi.object({
    includeCoaching: Joi.boolean().default(true),
    generateReport: Joi.boolean().default(false),
    privacyLevel: Joi.string().valid('HIGH', 'MEDIUM', 'LOW').default('HIGH'),
  }).required(),
});

const validateCreateAnalysis = (req: Request, res: Response, next: NextFunction): void => {
  const { error } = createAnalysisSchema.validate(req.body);
  if (error) {
    res.status(400).json({
      success: false,
      error: {
        code: 'VALIDATION_ERROR',
        message: 'Invalid request data',
        details: error.details,
      },
    });
    return;
  }
  next();
};

// ❌ DON'T
const createAnalysis = (req: Request, res: Response): void => {
  const { athleteId, videoUrl } = req.body; // No validation
  // Process request
};
```

**Security Validation**: Always validate for security

typescript

```typescript
// ✅ DO
const sanitizeInput = (input: string): string => {
  return input
    .replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi, '')
    .replace(/[<>]/g, '')
    .trim();
};

const validateAthleteAccess = async (
  athleteId: string,
  requesterId: string,
  role: UserRole
): Promise<boolean> => {
  // Verify requester has permission to access athlete data
  if (role === UserRole.ATHLETE && athleteId !== requesterId) {
    return false;
  }
  if (role === UserRole.COACH) {
    return await verifyCoachAthleteRelationship(requesterId, athleteId);
  }
  return true;
};

// ❌ DON'T
const getAthleteData = (req: Request, res: Response): void => {
  const athleteId = req.params.athleteId; // No access control
  // Return data without permission check
};
```

# 6. AI/ML Code Standards

## 6.1 Model Implementation

**Type Safety for ML Operations**:

typescript

```typescript
// ✅ DO
interface PoseDetectionResult {
  keypoints: Array<{
    name: string;
    position: { x: number; y: number };
    confidence: number;
  }>;
  boundingBox: {
    x: number;
    y: number;
    width: number;
    height: number;
  };
  confidence: number;
}

interface BiomechanicalFeatures {
  jointAngles: Record<string, number>;
  velocities: Record<string, number>;
  accelerations: Record<string, number>;
  symmetryIndex: number;
}

const extractBiomechanicalFeatures = (
  poses: PoseDetectionResult[]
): BiomechanicalFeatures => {
  // Type-safe feature extraction
};

// ❌ DON'T
const extractFeatures = (poses: any): any => {
  // Untyped ML operations
};
```

**Model Validation**:

typescript

```typescript
// ✅ DO
interface ModelValidationResult {
  isValid: boolean;
  confidence: number;
  warnings: string[];
  requiredConfidenceThreshold: number;
}

const validateAnalysisResult = (
  result: AnalysisResult,
  sport: SportType
): ModelValidationResult => {
  const warnings: string[] = [];
  let confidence = result.confidence;

  // Sport-specific validation
  if (sport === SportType.ShotPut && result.releaseAngle > 60) {
    warnings.push('Unusually high release angle detected');
    confidence *= 0.8;
  }

  if (confidence < MINIMUM_CONFIDENCE_THRESHOLD) {
    warnings.push('Low confidence analysis - manual review recommended');
  }

  return {
    isValid: confidence >= MINIMUM_CONFIDENCE_THRESHOLD,
    confidence,
    warnings,
    requiredConfidenceThreshold: MINIMUM_CONFIDENCE_THRESHOLD,
  };
};

// ❌ DON'T
const validateResult = (result: any): boolean => {
  return result.confidence > 0.5; // Magic numbers, no context
};
```

## 6.2 Privacy Protection in ML

**Automatic PII Removal**:

typescript

```
// ✅ DO
interface PrivacyConfig {
  blurFaces: boolean;
  removeBiometrics: boolean;
  anonymizeMovement: boolean;
  dataRetentionDays: number;
}

const applyPrivacyProtection = async (
  videoBuffer: Buffer,
  config: PrivacyConfig
): Promise<Buffer> => {
  let processedVideo = videoBuffer;

  if (config.blurFaces) {
    processedVideo = await blurDetectedFaces(processedVideo);
  }

  if (config.removeBiometrics) {
    processedVideo = await removeBiometricIdentifiers(processedVideo);
  }

  // Log privacy actions for compliance
  logger.info('Privacy protection applied', {
    actions: Object.keys(config).filter(key => config[key as keyof PrivacyConfig]),
    videoId: generateVideoHash(videoBuffer),
  });

  return processedVideo;
};

// ❌ DON'T
const processVideo = async (video: any): Promise<any> => {
  // No privacy protection
  return await analyzeVideo(video);
};
```

# 7. Testing Standards

## 7.1 Unit Testing

**Test Structure**: Use descriptive test names and AAA pattern

typescript

```typescript
// ✅ DO
describe('PowerQuotientCalculator', () => {
  describe('calculatePQS', () => {
    it('should return score between 0 and 100 for valid biomechanical data', () => {
      // Arrange
      const validBiomechanics: BiomechanicalData = {
        releaseAngle: 42,
        releaseVelocity: 12.5,
        powerTransfer: 0.85,
        footwork: { timing: 0.95, balance: 0.88 },
      };

      // Act
      const score = PowerQuotientCalculator.calculatePQS(validBiomechanics);

      // Assert
      expect(score).toBeGreaterThanOrEqual(0);
      expect(score).toBeLessThanOrEqual(100);
      expect(typeof score).toBe('number');
    });

    it('should throw ValidationError for biomechanical data with negative values', () => {
      // Arrange
      const invalidBiomechanics: BiomechanicalData = {
        releaseAngle: -10, // Invalid negative angle
        releaseVelocity: 12.5,
        powerTransfer: 0.85,
        footwork: { timing: 0.95, balance: 0.88 },
      };
```