Khel-Connect: API Documentation

1. Introduction

This document provides a detailed overview of the API and data model for the Khel-Connect platform. Given our serverless, Firebase-centric architecture, the "API" is not a set of traditional HTTP endpoints but a series of direct interactions with Firebase services (Authentication, Firestore, and Storage) via their respective SDKs.

All communication is handled securely by Firebase's built-in SDKs and their associated security rules, ensuring data integrity and user privacy.

2. Authentication

2.1. Athlete and Scout Authentication

Authentication for both athletes and sports authorities is handled by **Firebase Authentication**.

- **Endpoint**: Not applicable; authentication is handled via the Firebase SDK.
- **Method**: SDK function calls.

SDK Call	Description
firebase.auth().signInWithEmailAndPassword(email, password)	Authenticates a user with an email and password.
firebase.auth().signOut()	Signs out the currently authenticated user.
firebase.auth().onAuthStateChanged(user)	Listener that triggers whenever the user's sign-in state changes.

2.2. Data Security

Access to data is governed by Firebase Security Rules.

- **Athletes**: Can only read and write to their own profile data and upload videos to their designated folder in Firebase Storage.
- **Scouts**: Have read-only access to all player profiles and video metadata.

3. Storage API (Video Upload)

Video uploads are managed by **Firebase Storage**.

- **Endpoint**: Not applicable; upload is handled via the Firebase SDK.

- **Method**: SDK function call.

SDK Call	Description
firebase.storage().ref(path).put(file)	Uploads a video file to a specified path in Firebase Storage. The path is constructed to be unique for each user and video.
firebase.storage().ref(path).getDownloadURL()	Retrieves a public URL for the uploaded video, which is then stored in Firestore.

4. Firestore Database API

The **Firestore Database** is the central hub for all application data. Data is accessed via the SDK, and our data model is designed to be efficient for both read and write operations.

4.1. Data Models

users Collection

- **Path**: /users/{uid}
- **Description**: Stores profile information for both athletes and scouts.
- Document Fields:

Field	Type	Description
name	string	The user's full name.
email	string	The user's email address.
location	string	The user's location (e.g., city, state).
sport	string	The primary sport of the athlete (e.g., "Cricket").
role	string	The user's role: "athlete" or "scout".

videos Collection

- **Path**: /videos/{documentId}
- **Description**: Stores metadata and AI analysis results for each video clip uploaded by an athlete.
- Document Fields:

Field	Type	Description
userId	string	The uid of the user who uploaded the video.
videoUrl	string	The public download URL for the video in Firebase Storage.
uploadTimestamp	timestamp	The time the video was uploaded.

status	string	The processing status: "processing", "complete", or "failed".
skillScore	number	The overall numerical score (e.g., 88) generated by the AI model.
analysisMetrics	map	A map containing a detailed breakdown of the AI analysis, for example: { "armAngle": 90, "swingSpeed": 75 }.

4.2. Core Data Operations

These are the primary SDK calls used to interact with Firestore collections.

SDK Call	Path	Description
firebase.firestore().collection('videos').add(data)	videos	Creates a new video document after a video is uploaded to storage.
firebase.firestore().doc('videos/{docId}').update(data)	videos	Updates the status and AI analysis results of a video document.
firebase.firestore().collection('videos').where('userId', '==', uid).get()	videos	Retrieves all video documents for a specific user.
firebase.firestore().collection('videos').orderBy('skillScore', 'desc').get()	videos	Retrieves a leaderboard of all videos, ordered by skillScore. This query may require an index in Firestore.
firebase.firestore().doc('users/{uid}').get()	users	Retrieves a user's profile information.
firebase.firestore().collection('videos').onSnapshot(snapshot)	videos	A real-time listener to get updates on video processing status and results.

5. Backend Triggers (Cloud Functions)

The AI analysis pipeline is triggered automatically by a Firebase Cloud Function. This function does not have a publicly exposed API endpoint; it is an internal part of the application's backend.

- **Trigger**: onFinalize event on a new file creation in Firebase Storage.

- **Input**: The event object containing the file metadata.
- **Output**: A write operation to a Firestore document.

This architecture ensures that the complex video processing logic is completely abstracted away from the frontend, making the application more secure and maintainable.