# Requirements Document: FormFlux

Version: 2.0 (Web-First, Detailed Flows)
Date: June 15, 2025

## 1.0 Introduction

This document provides a detailed breakdown of the functional and system requirements for Project FormFlux. The primary goal is to build a web application that acts as a real-time AI fitness coach. This document outlines the specific interactions between the user, the React.js frontend, the Vercel backend, Firebase services, and the Google Gemini API for each feature phase.

## 2.0 General System & UI Requirements

- GR-1: User Interface (UI): The UI must be simple, clean, and modern, utilizing a minimalist aesthetic to keep the user focused. It should incorporate smooth transitions and subtle animations to feel responsive and high-quality.
- GR-2: Responsiveness: The entire web application must be fully responsive and provide a seamless experience on all common screen sizes, from mobile web browsers to widescreen desktop monitors.
- GR-3: Security: The Google API Key must be stored exclusively on the Vercel backend as an environment variable and never exposed to the client-side React application. All communication with the backend from authenticated users must be validated.
- GR-4: Performance: The end-to-end latency for AI audio feedback (from user action to audible response) must be optimized for a near-real-time feel, targeting under 500ms.

---

## PHASE 1: Version 1.0 (MVP) - The Core Web Experience

The objective of V1 is to launch the core product on the web to validate the AI coaching loop and user progress tracking.

### FR-1.0: User Authentication (Google Only)

- User Story: "As a new user, I want to create an account or log in instantly and securely using my Google account, without needing to create another password."

- Functional Requirements:
    1. The authentication page will present a single, prominent "Sign in with Google" button.
    2. There will be no email/password sign-up fields.

3. Upon successful authentication, the user is redirected to the main application dashboard.

- System Flow & Technical Implementation:
    1. Client (React): A user on the /login page clicks the "Sign in with Google" button.
    2. Client (React): The application calls the Firebase SDK's signInWithPopup(auth, googleProvider) function.
    3. Firebase: Firebase handles the entire Google OAuth 2.0 flow, presenting the Google account selection pop-up to the user.
    4. Client (React): On successful login, the Firebase SDK returns a user object. The application checks getAdditionalUserInfo(result).isNewUser.
    5. Client & Firestore: If the user is new, the client creates a new document in the users collection in Firestore with the user.uid as the document ID. This initial document will contain basic information like email, displayName, and createdAt.
    6. Client (React): The user's authentication state is managed globally (e.g., via React Context), and the user is navigated to the main dashboard (/dashboard).

### FR-2.0: Live Workout Session

- User Story: "As a logged-in user, I want to select an exercise, see myself on camera, and receive immediate, audible coaching on my form and reps."

- Functional Requirements:
    1. The dashboard will display a list of available exercises (e.g., "Squats," "Push-ups").
    2. Clicking an exercise starts the session. The browser must prompt for camera and microphone access if not already granted.
    3. The workout screen prominently displays the user's camera feed.
    4. An "End Set" button must be clearly visible.
    5. AI-generated audio provides all feedback (no visuals in V1).

- System Flow & Technical Implementation:
    1. Client (React): A logged-in user clicks "Start Squats." The app navigates to /workout/squats.
    2. Client (React): The app requests camera/mic access via navigator.mediaDevices.getUserMedia(). The video stream is rendered in a <video> element.
    3. Client (React): The app establishes a WebSocket connection to the Vercel backend. In the initial handshake, it sends the user's Firebase Auth ID Token for server-side verification.
    4. Backend (Vercel): The backend receives the WebSocket connection request. It uses the Firebase Admin SDK to verify the ID token. If valid, the connection is accepted.

5. Backend (Vercel): A new Gemini Live API session is initiated. The systemInstruction is set based on the exercise (e.g., "You are an expert squat coach..."). The responseModalities are set to [Modality.AUDIO].
6. The Live Loop:
    - a. Client: Captures frames from the video stream and audio from the mic, sending them as chunks over the WebSocket.
    - b. Backend: Relays these chunks directly to the Gemini Live API session.
    - c. Gemini: Analyzes the stream and sends back audio data (base64 encoded PCM).
    - d. Backend: Relays the audio data back to the client over the WebSocket.
    - e. Client: Receives the audio chunks, decodes them, and plays them through the browser's Web Audio API for seamless playback.

### FR-3.0: Post-Workout Summary & Charting

- User Story: "After I finish my set, I want to see a summary of how I performed and view a simple chart of my progress over time."

- Functional Requirements:
    1. When the "End Set" button is clicked, the live session terminates.
    2. The user is navigated to a summary page.
    3. This page displays the current session's: (1) Text Summary, (2) Form Rating /10, (3) Rep Count.
    4. The page also displays a simple line chart showing the formRating and repCount for all past sessions of that specific exercise.

- System Flow & Technical Implementation:
    1. Client (React): The user clicks "End Set." The app sends a final end_session message over the WebSocket and then closes the connection.
    2. Backend (Vercel): Upon receiving end_session, the backend sends a final prompt to the active Gemini session, asking it to return a JSON object with the summary: {"summaryText": "...", "formRating": 8, "repCount": 12}.
    3. Backend & Firestore: The backend receives the JSON summary. It then creates a new document in the Firestore collection users/{uid}/exercises/{exerciseName}/sessions/. This document includes the summary data plus a timestamp.
    4. Backend & Firestore: The backend then queries the same collection to retrieve all session documents, ordered by timestamp.
    5. Backend (Vercel): The backend sends a single HTTPS response (e.g., to a GET /api/summary/... request from the client) containing both the current session's summary and an array of all historical sessions.
    6. Client (React): The app receives this data. It navigates to the /summary/{exerciseName} page, displays the current session's stats, and passes the historical data array to a charting library (e.g., Recharts) to render the progress graphs.

---

## PHASE 2: Version 2.0 - Enhanced Web Coaching

### FR-4.0: Visual Feedback & Audio Control

- User Story: "As a user, I want the option to work out silently, receiving only visual cues on screen to help me correct my form, like an arrow showing me to lower my hips."

- Functional Requirements:
    1. A clear "Audio On/Off" toggle switch is available on the workout screen.
    2. When audio is ON, the system functions as in V1.
    3. When audio is OFF, no sound is played. Instead, visual overlays (e.g., arrows, highlights) appear on the screen as needed.

- System Flow & Technical Implementation:
    1. Client (React): The state of the audio toggle (isAudioEnabled) is managed in the component.
    2. Client (React): When establishing the WebSocket connection, this preference is sent in the initial message: { authToken: "...", exercise: "squats", audioEnabled: false }.
    3. Backend (Vercel): When initiating the Gemini Live API session, the backend checks the audioEnabled flag.
        - If true, responseModalities is set to [Modality.AUDIO].
        - If false, responseModalities is set to [Modality.TEXT].
    4. Backend (Vercel): The systemInstruction will be updated to instruct the AI to use tool_calls for visual feedback when it identifies a form error (e.g., display_correction_arrow).
    5. The Live Loop (Audio OFF):
        - Gemini sends back toolCall messages instead of audio.
        - The backend relays these toolCall JSON objects to the client.
        - The client receives the command (e.g., {name: 'display_correction_arrow', args: {direction: 'down', bodyPart: 'hips'}}) and renders the appropriate visual overlay on the screen for a few seconds.

---

## PHASE 3: Version 3.0 - Personalization, Mobile & Monetization

### FR-5.0: User Onboarding for Personalization

- User Story: "As a new user, I want the app to ask about my fitness goals and experience so the AI's coaching feels more tailored to me."

- Functional Requirements:

1. Immediately after a new user's first sign-in, a mandatory onboarding flow is triggered.
2. This flow will ask questions like "What is your fitness level? (Beginner/Intermediate/Advanced)" and "Do you have any areas of sensitivity? (e.g., Knees, Lower Back)".
3. The answers are saved to the user's profile.

- System Flow & Technical Implementation:
    1. Client & Firestore: After first sign-in, the client checks for an onboardingComplete: true flag in the users/{uid} Firestore document.
    2. Client (React): If the flag is missing or false, it renders a multi-step onboarding modal.
    3. Client & Firestore: Upon completion, the client writes the collected answers to the user's document in Firestore and sets onboardingComplete: true.
    4. Usage: When a user starts any future workout session, the client first fetches this onboarding data from Firestore.
    5. Client (React): This personalization data is sent in the initial WebSocket message to the backend.
    6. Backend (Vercel): The backend dynamically prepends this information to the systemInstruction for the Gemini session. (e.g., "You are coaching a user who is a beginner with sensitive knees. Your feedback must be encouraging and prioritize safety over intensity...").

### FR-6.0: Custom Voices

- User Story: “I want to be able to control and have options for the voice of the assistant that’s speaking to me”

- Functional Requirements:
    1. In an assistant button on the top or a settings page, there should be a way to see all the available voices/personalities
    2. User should be able to click these personalities and it should play the audio to show user what it sounds like
    3. There should be a brief description of each personal trainer.
    4. When the user selects a new voice, then on any future workout session the client should use this voice.

- The user will be able to select from a list of assistants each which have a different voice (ie the various gemini voices that are available)

### FR-7.0: React Native App & Monetization

- This phase involves porting the proven V2 web features to a React Native mobile application and introducing subscription tiers with usage limits enforced by a payment provider like Stripe. The detailed requirements for this phase will be drafted upon successful completion and validation of V1 and V2.