Modulo de Asistente Fitness

FitTrack

Sistema Inteligente de Asistencia Nutricional y Fitness

Este modulo proporciona un asistente inteligente basado en IA que combina analisis nutricional de imagenes, chat personalizado y recomendaciones fitness integradas con el progreso del usuario.

Versión 1.0

14 de octubre de 2025

Índice

1 Introduccion al Modulo de Asistente Fitness

El modulo de Asistente Fitness es uno de los componentes mas avanzados y tecnologicamente sofisticados de FitTrack. Integra inteligencia artificial de Google Gemini para proporcionar asistencia nutricional personalizada, analisis de imagenes de comida y recomendaciones fitness basadas en el progreso real del usuario.

1.1 Caracteristicas Principales

- **Chat Inteligente**: Asistente conversacional especializado en nutricion y fitness
- **Analisis de Imagenes**: Reconocimiento automatico de comidas y calculo de valores nutricionales
- **Personalizacion Avanzada**: Recomendaciones basadas en perfil, objetivos y progreso del usuario
- **Integracion Completa**: Acceso a datos de gimnasio, running y metricas de salud
- **IA de Ultima Generacion**: Utiliza Google Gemini 2.5 Flash para analisis preciso
- **Interfaz Dual**: Chat conversacional y analizador de imagenes
- **Calculos Automaticos**: BMR, TDEE, necesidades proteicas personalizadas

1.2 Arquitectura General

El modulo sigue una arquitectura de microservicios con separacion clara de responsabilidades:

1. **Capa de Presentacion**: Componentes React con TypeScript
2. **Capa de API**: Endpoints especializados para chat y analisis de imagenes
3. **Capa de IA**: Integracion con Google Gemini para procesamiento inteligente
4. **Capa de Datos**: Acceso a perfil de usuario y datos de actividad
5. **Capa de Servicios**: Calculos nutricionales y recomendaciones personalizadas

2 Estructura de Archivos

2.1 Organizacion del Modulo

```
app/
+-- meals/                    # Pagina principal del asistente
|   +-- page.tsx              # Componente principal con tabs
+-- api/                      # APIs del asistente
|   +-- chat/                 # API de chat conversacional
|   |   +-- route.ts          # Endpoint principal de chat
|   +-- analyze/              # API de analisis de imagenes
|   |   +-- route.ts          # Endpoint de analisis nutricional
```

```
+-- components/meals/        # Componentes del asistente
|   +-- chat-interface.tsx   # Interfaz de chat
|   +-- image-analyzer.tsx   # Analizador de imagenes
```

Listing 1: Estructura de archivos del Asistente Fitness

3 APIs y Endpoints

3.1 API de Chat - /api/chat

El endpoint principal que maneja tanto el chat conversacional como el analisis de imagenes.

3.1.1 Estructura del Endpoint

```
import { generateText, generateObject } from "ai"
import { google } from "@ai-sdk/google"
import { type NextRequest, NextResponse } from "next/server"
import { createClient } from "@/lib/supabase/server"
import { z } from "zod"

// Schema para analisis de imagenes
const foodAnalysisSchema = z.object({
  foodName: z.string().describe("Nombre del plato o comida"),
  calories: z.number().describe("Numero estimado de calorias"),
  protein: z.number().describe("Gramos de proteina"),
  carbs: z.number().describe("Gramos de carbohidratos"),
  fats: z.number().describe("Gramos de grasas"),
  fiber: z.number().optional().describe("Gramos de fibra"),
  serving: z.string().describe("Descripcion del tamano de la porcion"),
  ingredients: z.array(z.string()).describe("Lista de ingredientes
    visibles"),
  recommendations: z.string().describe("Breve recomendacion nutricional
    "),
  confidence: z.enum(["alta", "media", "baja"]).describe("Nivel de
    confianza en el analisis"),
})

export async function POST(request: NextRequest) {
  // Manejo de autenticacion y validacion
  // Enrutamiento a chat o analisis de imagen
}
```

Listing 2: Estructura basica del endpoint /api/chat

3.1.2 Manejo de Chat Conversacional

```
async function handleChatMessage(message: string, userProfile: any) {
  // 1. Recopilacion de datos del usuario
  let exerciseData = ""
  try {
    const exercises = await getUniqueExercises()
    if (exercises.length > 0) {
```

```
      exerciseData = `\n\nTus ejercicios registrados: ${exercises.join(", ")}`

      const lastExercise = exercises[0]
      const history = await getExerciseHistory(lastExercise, 30)
      if (history.length > 0) {
        const latest = history[0]
        exerciseData += `\n\nUltimo registro de ${lastExercise}: ${latest.weight_kg}kg x ${latest.repetitions} reps`
      }
    }
  } catch (error) {
    console.log("No se pudo obtener historial de ejercicios")
  }

  // 2. Datos de running
  let runningData = ""
  try {
    const runningSessions = await getRunningHistory(30)
    if (runningSessions.length > 0) {
      const totalDistance = runningSessions.reduce((sum, session) => sum + session.distance, 0)
      const avgPace = runningSessions.reduce((sum, session) => sum + session.pace, 0) / runningSessions.length
      const lastSession = runningSessions[0]

      runningData = `\n\nDATOS DE RUNNING (ultimos 30 dias):
- Total de sesiones: ${runningSessions.length}
- Distancia total: ${totalDistance.toFixed(2)}km
- Pace promedio: ${avgPace.toFixed(2)} min/km
- Ultima sesion: ${lastSession.distance}km en ${lastSession.duration} minutos (${lastSession.pace.toFixed(2)} min/km)`
    }
  } catch (error) {
    console.log("No se pudo obtener historial de running")
  }

  // 3. Datos de gimnasio
  let gymData = ""
  try {
    const workouts = await getWorkouts()
    if (workouts.length > 0) {
      const recentWorkouts = workouts.slice(0, 5)
      gymData = `\n\nULTIMOS ENTRENAMIENTOS EN GIMNASIO:`
      recentWorkouts.forEach((workout: any) => {
        const date = new Date(workout.created_at).toLocaleDateString()
        gymData += `\n- ${workout.exercise_name}: ${workout.weight_kg || 0}kg x ${workout.repetitions || 0} reps x ${workout.sets || 0} sets (${date})`
      })
    }
  } catch (error) {
    console.log("No se pudo obtener workouts del gimnasio")
  }

  // 4. Calculos nutricionales personalizados
  let bmr = 0, tdee = 0, proteinMin = 0, proteinMax = 0

```

```
58   if (userProfile?.weight && userProfile?.height && userProfile?.age &&
      userProfile?.sex) {
59     if (userProfile.sex === "male") {
60       bmr = 10 * userProfile.weight + 6.25 * userProfile.height - 5 *
     userProfile.age + 5
61     } else {
62       bmr = 10 * userProfile.weight + 6.25 * userProfile.height - 5 *
     userProfile.age - 161
63     }
64     tdee = Math.round(bmr * 1.55)
65     proteinMin = Math.round(userProfile.weight * 1.6)
66     proteinMax = Math.round(userProfile.weight * 2.2)
67   }
68
69   // 5. Generacion de prompt personalizado
70   const model = google("gemini-2.5-flash")
71
72   const prompt = `Eres un asistente nutricional experto especializado en
       fitness y salud. Tu nombre es "Asistente Nutricional de FitTrack".
73
74 INFORMACION DEL USUARIO:
75 ${userProfile?.weight ? `- Peso: ${userProfile.weight}kg` : ""}
76 ${userProfile?.height ? `- Altura: ${userProfile.height}cm` : ""}
77 ${userProfile?.age ? `- Edad: ${userProfile.age} anos` : ""}
78 ${userProfile?.sex ? `- Sexo: ${userProfile.sex === "male" ? "Masculino"
       : "Femenino"}` : ""}
79 ${userProfile?.bmi ? `- IMC: ${userProfile.bmi} (${userProfile.
     bmiCategory})` : ""}
80 ${bmr > 0 ? `- Metabolismo basal (BMR): ${Math.round(bmr)} cal/dia` :
     ""}
81 ${tdee > 0 ? `- TDEE (actividad moderada): ${tdee} cal/dia` : ""}
82 ${proteinMin > 0 ? `- Proteina recomendada: ${proteinMin}-${proteinMax}g
     /dia` : ""}
83 ${exerciseData}
84 ${runningData}
85 ${gymData}
86
87 PREGUNTA DEL USUARIO: ${message}
88
89 INSTRUCCIONES:
90 - Responde en espanol de forma clara, concisa y util
91 - Usa formato markdown para mejor legibilidad
92 - Personaliza tu respuesta basandote en TODOS los datos del usuario
93 - Si el usuario pregunta sobre su progreso, analiza sus datos de
     ejercicios
94 - Proporciona informacion basada en evidencia cientifica
95 - Incluye ejemplos practicos y cantidades especificas
96 - Manten un tono motivador y profesional
97
98 Responde a la pregunta del usuario ahora:`
99
100  const { text } = await generateText({
101    model,
102    prompt,
103  })
104
105  return NextResponse.json({ response: text })
```

```
}
```

Listing 3: Funcion handleChatMessage - Chat personalizado

3.1.3 Manejo de Analisis de Imagenes

```
async function handleImageAnalysis(image: string) {
  try {
    // Remover prefijo data URL
    const base64Image = image.replace(/^data:image\/\w+;base64,/, "")

    const model = google("gemini-2.0-flash-exp")

    const prompt = `Analiza esta imagen de comida y proporciona la
   siguiente informacion en formato JSON:

{
  "foodName": "nombre del plato o comida",
  "calories": numero estimado de calorias,
  "protein": gramos de proteina,
  "carbs": gramos de carbohidratos,
  "fats": gramos de grasas,
  "fiber": gramos de fibra (opcional),
  "serving": "descripcion del tamano de la porcion (ej: '1 plato mediano
   ', '200g')",
  "ingredients": ["lista", "de", "ingredientes", "visibles"],
  "recommendations": "breve recomendacion nutricional o consejo sobre
   esta comida",
  "confidence": "alta/media/baja - tu nivel de confianza en el analisis"
}

Se lo mas preciso posible. Si no puedes identificar la comida claramente
   , indica baja confianza y proporciona tu mejor estimacion.`

    const { object } = await generateObject({
      model,
      schema: foodAnalysisSchema,
      prompt,
      messages: [
        {
          role: "user",
          content: [
            { type: "text", text: prompt },
            {
              type: "image",
              image: `${base64Image}`,
            },
          ],
        },
      ],
    })

    return NextResponse.json(object)
  } catch (error) {
    console.error("Error analyzing food image:", error)
    return NextResponse.json({ error: "Error al analizar la imagen. Por
   favor intenta de nuevo." }, { status: 500 })
```

```
  }
}
```

Listing 4: Funcion handleImageAnalysis - Analisis con IA

3.2 API de Analisis - /api/analyze

Endpoint especializado para el analisis detallado de imagenes de comida.

```
import { GoogleGenerativeAI } from "@google/generative-ai"
import { NextResponse } from "next/server"
import { createClient } from "@/lib/supabase/server"

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY || "
   AIzaSyDYVDyl_8rod_HvIKxgrUgLHMfMjPc5uNA")

export async function POST(request: Request) {
  try {
    const { image, userProfile } = await request.json()

    const supabase = await createClient()
    const { data: { user } } = await supabase.auth.getUser()

    if (!user) {
      return NextResponse.json({ error: "No autenticado" }, { status:
   401 })
    }

    // Prompt del sistema personalizado
    let systemPrompt = `Eres un experto nutricionista y analista de
   alimentos de FitTrack. Tu tarea es analizar imagenes de comidas y
   proporcionar informacion nutricional detallada y personalizada.

Cuando analices una imagen de comida, debes:
1. Identificar todos los alimentos visibles en la imagen
2. Estimar las porciones de cada alimento
3. Calcular aproximadamente las calorias totales
4. Desglosar macronutrientes (proteinas, carbohidratos, grasas)
5. Evaluar la calidad nutricional de la comida
6. Proporcionar recomendaciones personalizadas segun el perfil del
   usuario
7. Sugerir mejoras o alternativas mas saludables si es apropiado

Formato de respuesta:
- Se especifico y detallado
- Usa lenguaje claro y motivador
- Incluye numeros aproximados (calorias, gramos de macros)
- Personaliza segun el perfil del usuario (objetivos, IMC, etc.)
- Responde en espanol de forma natural

Si la imagen no contiene comida o no es clara, indicalo amablemente y
   pide una mejor imagen.`

    // Personalizacion segun perfil del usuario
    if (userProfile) {
      systemPrompt += `\n\nPerfil del usuario:`
      if (userProfile.weight) systemPrompt += `\n- Peso: ${userProfile.
   weight} kg`
```

```
43      if (userProfile.height) systemPrompt += `\n- Estatura: ${
    userProfile.height} cm`
44      if (userProfile.age) systemPrompt += `\n- Edad: ${userProfile.age}
     anos`
45      if (userProfile.sex) systemPrompt += `\n- Sexo: ${userProfile.sex
    }`
46      if (userProfile.bmi) systemPrompt += `\n- IMC: ${userProfile.bmi}
    (${userProfile.bmiCategory})`
47
48      // Calculo de BMR personalizado
49      if (userProfile.weight && userProfile.height && userProfile.age &&
     userProfile.sex) {
50        let bmr: number
51        if (userProfile.sex === "male") {
52          bmr = 10 * userProfile.weight + 6.25 * userProfile.height - 5
    * userProfile.age + 5
53        } else {
54          bmr = 10 * userProfile.weight + 6.25 * userProfile.height - 5
    * userProfile.age - 161
55        }
56
57        systemPrompt += `\n- Metabolismo basal (BMR): ${Math.round(bmr)}
     cal/dia`
58        systemPrompt += `\n\nUsa esta informacion para personalizar tus
    recomendaciones y evaluar si esta comida se ajusta a sus necesidades
    .`
59      }
60    }
61
62    const model = genAI.getGenerativeModel({ model: "gemini-2.5-flash"
    })
63
64    // Procesamiento de imagen
65    const base64Image = image.replace(/^data:image\/\w+;base64,/, "")
66
67    const result = await model.generateContent([
68      systemPrompt,
69      {
70        inlineData: {
71          data: base64Image,
72          mimeType: "image/jpeg",
73        },
74      },
75    ])
76
77    const response = await result.response
78    const text = response.text()
79
80    return NextResponse.json({ analysis: text })
81  } catch (error) {
82    console.error("Error analyzing image:", error)
83    return NextResponse.json({ error: "Error al analizar la imagen. Por
    favor intenta de nuevo." }, { status: 500 })
84  }
85 }
```

Listing 5: API de analisis de imagenes especializada

4 Componentes React

4.1 Pagina Principal - MealsPage

El componente principal que coordina toda la funcionalidad del asistente fitness.

```
import ChatInterface from "@/components/meals/chat-interface"
import ImageAnalyzer from "@/components/meals/image-analyzer"
import { MessageCircle, ArrowLeft, Camera } from "lucide-react"
import Link from "next/link"
import { Button } from "@/components/ui/button"
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/
    ui/tabs"

export default function MealsPage() {
  return (
    <div className="min-h-screen bg-gradient-to-br from-gray-100 to-gray
    -200 dark:from-gray-800 dark:to-gray-900">
      <div className="container mx-auto px-4 py-8">
        {/* Header */}
        <div className="mb-8">
          <div className="flex items-center gap-4 mb-4">
            <Button variant="outline" size="sm" asChild>
              <Link href="/">
                <ArrowLeft className="h-4 w-4 mr-2" />
                Volver
              </Link>
            </Button>
          </div>
          <div className="text-center">
            <div className="flex items-center justify-center gap-3 mb
    -4">
              <MessageCircle className="h-10 w-10 text-orange-600" />
              <h1 className="text-3xl font-bold text-gray-900 dark:text-
    white">Asistente Nutricional</h1>
            </div>
            <p className="text-lg text-gray-600 dark:text-gray-300 max-w
    -2xl mx-auto">
              Obten consejos personalizados sobre alimentacion, analiza
    tus comidas con IA y planifica tu nutricion
            </p>
          </div>
        </div>

        <Tabs defaultValue="chat" className="w-full">
          <TabsList className="grid w-full max-w-md mx-auto grid-cols-2
    mb-8">
            <TabsTrigger value="chat" className="flex items-center gap
    -2">
              <MessageCircle className="h-4 w-4" />
              Chat con IA
            </TabsTrigger>
            <TabsTrigger value="image" className="flex items-center gap
    -2">
              <Camera className="h-4 w-4" />
              Analizar Comida
            </TabsTrigger>
          </TabsList>
```

```
          <TabsContent value="chat">
            <ChatInterface />
          </TabsContent>
          <TabsContent value="image">
            <ImageAnalyzer/>
          </TabsContent>
        </Tabs>
      </div>
    </div>
  )
}
```

Listing 6: app/meals/page.tsx - Estructura completa

Caracteristicas del componente:

- **Interfaz de Tabs**: Navegacion entre chat y analisis de imagenes
- **Header Informativo**: Descripcion clara de las funcionalidades
- **Diseño Responsivo**: Adaptable a diferentes tamanos de pantalla
- **Tema Oscuro**: Soporte completo para modo oscuro
- **Navegacion**: Boton de regreso al dashboard principal

4.2 Componente ChatInterface

Interfaz de chat conversacional con el asistente nutricional.

```
"use client"

import type React from "react"
import { useState, useRef, useEffect } from "react"
import { Button } from "@/components/ui/button"
import { Input } from "@/components/ui/input"
import { Card, CardContent, CardHeader, CardTitle } from "@/components/
    ui/card"
import { Badge } from "@/components/ui/badge"
import { Send, MessageCircle, Utensils, Apple, Coffee, Calculator, User
    } from "lucide-react"
import { getUserProfile } from "@/lib/user-actions"
import { calculateBMI } from "@/lib/health-actions"

interface Message {
  id: string
  content: string
  isUser: boolean
  timestamp: Date
}

interface UserProfile {
  weight: number | null
  height: number | null
  dateOfBirth: string | null
  sex: string | null
  bmi?: number
  bmiCategory?: string
  age?: number
```

```
}

export default function ChatInterface() {
  const messageIdCounter = useRef(0)
  const [messages, setMessages] = useState<Message[]>([])
  const [inputValue, setInputValue] = useState("")
  const [isLoading, setIsLoading] = useState(false)
  const [isClient, setIsClient] = useState(false)
  const [userProfile, setUserProfile] = useState<UserProfile | null>(
   null)
  const messagesEndRef = useRef<HTMLDivElement>(null)

  // Funciones de utilidad
  const scrollToBottom = () => {
    messagesEndRef.current?.scrollIntoView({ behavior: "smooth" })
  }

  useEffect(() => {
    scrollToBottom()
  }, [messages])

  useEffect(() => {
    setIsClient(true)
    loadUserProfile()
    setMessages([
      {
        id: (++messageIdCounter.current).toString(),
        content: "Hola! Soy tu asistente nutricional de FitTrack. Puedo
   ayudarte con consejos sobre alimentacion, calcular tus necesidades
   caloricas, sugerir recetas y crear planes de comidas. Preguntame
   sobre: calorias, proteinas, ganar musculo, perder peso, meal prep,
   recetas, suplementos, y mucho mas. En que puedo ayudarte?",
        isUser: false,
        timestamp: new Date(),
      },
    ])
  }, [])

  const loadUserProfile = async () => {
    try {
      const profile = await getUserProfile()
      if (profile) {
        let age: number | undefined
        let bmi: number | undefined
        let bmiCategory: string | undefined

        if (profile.dateOfBirth) {
          const birthDate = new Date(profile.dateOfBirth)
          age = new Date().getFullYear() - birthDate.getFullYear()
        }

        if (profile.weight && profile.height) {
          const bmiResult = await calculateBMI(profile.weight, profile.
   height, age, profile.sex || undefined)
          bmi = bmiResult.bmi
          bmiCategory = bmiResult.category
        }

```

```
        setUserProfile({
          weight: profile.weight,
          height: profile.height,
          dateOfBirth: profile.dateOfBirth,
          sex: profile.sex,
          bmi,
          bmiCategory,
          age,
        })
      }
    } catch (error) {
      console.error("Error loading user profile:", error)
    }
  }

  const handleSendMessage = async () => {
    if (!inputValue.trim()) return

    const userMessage: Message = {
      id: (++messageIdCounter.current).toString(),
      content: inputValue,
      isUser: true,
      timestamp: new Date(),
    }

    setMessages((prev) => [...prev, userMessage])
    setInputValue("")
    setIsLoading(true)

    try {
      const response = await fetch("/api/chat", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          message: inputValue,
          userProfile,
        }),
      })

      const data = await response.json()

      if (data.error) {
        throw new Error(data.error)
      }

      const aiResponse: Message = {
        id: (++messageIdCounter.current).toString(),
        content: data.response.replaceAll("**", ""),
        isUser: false,
        timestamp: new Date(),
      }

      setMessages((prev) => [...prev, aiResponse])
    } catch (error) {
      console.error("Error sending message:", error)
      const errorMessage: Message = {
```

```
138         id: (++messageIdCounter.current).toString(),
139         content: `Error: ${error instanceof Error ? error.message : "Lo
    siento, hubo un error al procesar tu mensaje. Por favor intenta de
    nuevo o verifica tu conexion."}\n\nSi el error menciona "API key",
    necesitas crear un archivo .env.local con tu clave de Gemini.
    Consulta CONFIGURACION_API.md para mas detalles.`,
140         isUser: false,
141         timestamp: new Date(),
142       }
143       setMessages((prev) => [...prev, errorMessage])
144     } finally {
145       setIsLoading(false)
146     }
147   }
148
149   const handleKeyPress = (e: React.KeyboardEvent) => {
150     if (e.key === "Enter" && !e.shiftKey) {
151       e.preventDefault()
152       handleSendMessage()
153     }
154   }
155
156   // Resto del componente...
157 }
```

Listing 7: components/meals/chat-interface.tsx - Estructura principal

Caracteristicas del chat:

- **Estado de Mensajes**: Gestiona historial completo de conversacion
- **Perfil de Usuario**: Carga y utiliza datos personales para personalizacion
- **Temas Rapidos**: Badges clickeables para consultas comunes
- **Indicadores de Carga**: Feedback visual durante procesamiento
- **Manejo de Errores**: Mensajes informativos para problemas de API
- **Scroll Automatico**: Navegacion fluida en conversaciones largas

4.3 Componente ImageAnalyzer

Analizador de imagenes de comida con IA.

```
1 "use client"
2
3 import type React from "react"
4 import { useState, useRef, useEffect } from "react"
5 import { Button } from "@/components/ui/button"
6 import { Card, CardContent, CardHeader, CardTitle } from "@/components/
    ui/card"
7 import { Badge } from "@/components/ui/badge"
8 import { Camera, Upload, X, Sparkles, User, Loader2 } from "lucide-react
    "
9 import { getUserProfile } from "@/lib/user-actions"
10 import { calculateBMI } from "@/lib/health-actions"
11
```

```
interface UserProfile {
  weight: number | null
  height: number | null
  dateOfBirth: string | null
  sex: string | null
  bmi?: number
  bmiCategory?: string
  age?: number
}

export default function ImageAnalyzer() {
  const [selectedImage, setSelectedImage] = useState<string | null>(null
  )
  const [analysis, setAnalysis] = useState<string | null>(null)
  const [isAnalyzing, setIsAnalyzing] = useState(false)
  const [isClient, setIsClient] = useState(false)
  const [userProfile, setUserProfile] = useState<UserProfile | null>(
  null)
  const fileInputRef = useRef<HTMLInputElement>(null)
  const cameraInputRef = useRef<HTMLInputElement>(null)

  useEffect(() => {
    setIsClient(true)
    loadUserProfile()
  }, [])

  const loadUserProfile = async () => {
    try {
      const profile = await getUserProfile()
      if (profile) {
        let age: number | undefined
        let bmi: number | undefined
        let bmiCategory: string | undefined

        if (profile.dateOfBirth) {
          const birthDate = new Date(profile.dateOfBirth)
          age = new Date().getFullYear() - birthDate.getFullYear()
        }

        if (profile.weight && profile.height) {
          const bmiResult = await calculateBMI(profile.weight, profile.
  height, age, profile.sex || undefined)
          bmi = bmiResult.bmi
          bmiCategory = bmiResult.category
        }

        setUserProfile({
          weight: profile.weight,
          height: profile.height,
          dateOfBirth: profile.dateOfBirth,
          sex: profile.sex,
          bmi,
          bmiCategory,
          age,
        })
      }
    } catch (error) {
      console.error("Error loading user profile:", error)
```

```
    }
  }

  const handleFileSelect = (event: React.ChangeEvent<HTMLInputElement>)
   => {
    const file = event.target.files?.[0]
    if (file) {
      const reader = new FileReader()
      reader.onloadend = () => {
        setSelectedImage(reader.result as string)
        setAnalysis(null)
      }
      reader.readAsDataURL(file)
    }
  }

  const handleAnalyze = async () => {
    if (!selectedImage) return

    setIsAnalyzing(true)
    setAnalysis(null)

    try {
      const response = await fetch("/api/analyze", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          image: selectedImage,
          userProfile,
        }),
      })

      const data: { error: string; analysis: string } = await response.
    json()

      if (data.error) {
        throw new Error(data.error)
      }

      setAnalysis(data.analysis.replaceAll("**", ""))
    } catch (error) {
      console.error("Error analyzing image:", error)
      setAnalysis("Lo siento, hubo un error al analizar la imagen. Por
    favor intenta de nuevo o verifica tu conexion.")
    } finally {
      setIsAnalyzing(false)
    }
  }

  const handleClear = () => {
    setSelectedImage(null)
    setAnalysis(null)
    if (fileInputRef.current) fileInputRef.current.value = ""
    if (cameraInputRef.current) cameraInputRef.current.value = ""
  }

```

```
122   // Resto del componente...
123 }
```

Listing 8: components/meals/image-analyzer.tsx - Estructura principal

Caracteristicas del analizador:

- **Captura de Imagen**: Soporte para camara y subida de archivos
- **Previsualizacion**: Vista previa de la imagen seleccionada
- **Analisis en Tiempo Real**: Procesamiento inmediato con IA
- **Resultados Detallados**: Informacion nutricional completa
- **Personalizacion**: Recomendaciones basadas en perfil del usuario
- **Manejo de Estados**: Carga, error y exito claramente diferenciados

5 Integracion con IA

5.1 Google Gemini Integration

El modulo utiliza Google Gemini 2.5 Flash para procesamiento de lenguaje natural y analisis de imagenes.

5.1.1 Configuracion de API

```
1 // Variables de entorno requeridas
2 GOOGLE_GENERATIVE_AI_API_KEY=tu_clave_de_gemini_aqui
3 GEMINI_API_KEY=tu_clave_de_gemini_aqui
4
5 // Importaciones necesarias
6 import { generateText, generateObject } from "ai"
7 import { google } from "@ai-sdk/google"
8 import { GoogleGenerativeAI } from "@google/generative-ai"
```

Listing 9: Configuracion de Google Gemini

5.1.2 Modelos Utilizados

- **Gemini 2.5 Flash**: Para chat conversacional y analisis de texto
- **Gemini 2.0 Flash Exp**: Para analisis de imagenes con schema estructurado
- **GoogleGenerativeAI**: Para analisis detallado de imagenes de comida

5.2 Schemas de Validacion

5.2.1 Schema de Analisis de Comida

```
const foodAnalysisSchema = z.object({
  foodName: z.string().describe("Nombre del plato o comida"),
  calories: z.number().describe("Numero estimado de calorias"),
  protein: z.number().describe("Gramos de proteina"),
  carbs: z.number().describe("Gramos de carbohidratos"),
  fats: z.number().describe("Gramos de grasas"),
  fiber: z.number().optional().describe("Gramos de fibra"),
  serving: z.string().describe("Descripcion del tamano de la porcion"),
  ingredients: z.array(z.string()).describe("Lista de ingredientes visibles"),
  recommendations: z.string().describe("Breve recomendacion nutricional"),
  confidence: z.enum(["alta", "media", "baja"]).describe("Nivel de confianza en el analisis"),
})
```

Listing 10: Schema Zod para validacion de analisis

6 Calculos Nutricionales

6.1 Metabolismo Basal (BMR)

Calculo del metabolismo basal utilizando la ecuacion de Mifflin-St Jeor.

```
// Ecuacion de Mifflin-St Jeor para BMR
if (userProfile.sex === "male") {
  bmr = 10 * userProfile.weight + 6.25 * userProfile.height - 5 * userProfile.age + 5
} else {
  bmr = 10 * userProfile.weight + 6.25 * userProfile.height - 5 * userProfile.age - 161
}

// TDEE (Total Daily Energy Expenditure) - Actividad moderada
tdee = Math.round(bmr * 1.55)
```

Listing 11: Calculo de BMR personalizado

6.2 Necesidades Proteicas

Calculo de necesidades proteicas basado en el peso corporal.

```
// Rango de proteinas para fitness (1.6-2.2g por kg de peso)
proteinMin = Math.round(userProfile.weight * 1.6)
proteinMax = Math.round(userProfile.weight * 2.2)
```

Listing 12: Calculo de proteinas recomendadas

7 Integracion con Datos del Usuario

7.1 Acceso a Datos de Actividad

El asistente integra datos de todos los modulos de FitTrack:

```
// Datos de ejercicios de gimnasio
const exercises = await getUniqueExercises()
const history = await getExerciseHistory(lastExercise, 30)

// Datos de running
const runningSessions = await getRunningHistory(30)
const totalDistance = runningSessions.reduce((sum, session) => sum +
    session.distance, 0)
const avgPace = runningSessions.reduce((sum, session) => sum + session.
    pace, 0) / runningSessions.length

// Datos de entrenamientos
const workouts = await getWorkouts()
const recentWorkouts = workouts.slice(0, 5)
```

Listing 13: Integracion con datos de actividad

7.2 Personalizacion Avanzada

El sistema utiliza el perfil completo del usuario para personalizar respuestas:

- **Datos Demograficos**: Peso, altura, edad, sexo
- **Metricas de Salud**: IMC, categoria de peso
- **Actividad Reciente**: Ejercicios, running, entrenamientos
- **Objetivos Fitness**: Basados en patrones de actividad
- **Progreso Historico**: Tendencias y mejoras

8 Flujos de Datos

8.1 Flujo de Chat Conversacional

1. **Usuario escribe mensaje** en ChatInterface
2. **Validacion de entrada** en cliente
3. **Envio a /api/chat** con perfil de usuario
4. **Recopilacion de datos** de gimnasio, running y salud
5. **Calculo de metricas** nutricionales personalizadas
6. **Generacion de prompt** contextualizado
7. **Procesamiento con Gemini** 2.5 Flash
8. **Respuesta personalizada** al usuario
9. **Actualizacion de UI** con mensaje del asistente

8.2 Flujo de Analisis de Imagenes

1. **Usuario selecciona imagen** (camara o archivo)
2. **Previsualizacion** de la imagen seleccionada
3. **Envio a /api/analyze** con imagen y perfil
4. **Procesamiento de imagen** con Gemini 2.5 Flash
5. **Analisis nutricional** detallado
6. **Recomendaciones personalizadas** basadas en perfil
7. **Presentacion de resultados** estructurados
8. **Opciones de accion** (nuevo analisis, limpiar)

9 Caracteristicas Avanzadas

9.1 Temas Rapidos

El chat incluye temas predefinidos para consultas comunes:

```
const quickTopics = [
  "Mi perfil",
  "Mis ejercicios",
  "Cuantas calorias necesito",
  "Ganar musculo",
  "Perder peso",
  "Pre entreno",
  "Post entreno",
  "Meal prep",
  "Recetas",
  "Suplementos",
  "Proteinas",
  "Carbohidratos"
]
```

Listing 14: Temas rapidos del chat

9.2 Indicadores de Confianza

El analisis de imagenes incluye niveles de confianza:

- **Alta**: Identificacion clara de alimentos y porciones
- **Media**: Identificacion parcial con estimaciones razonables
- **Baja**: Imagen poco clara o alimentos no identificables

9.3 Validacion de Imagenes

```
// Formatos soportados
accept="image/*"
capture="environment"  // Para camara trasera en moviles

// Validacion en cliente
const file = event.target.files?.[0]
if (file) {
  const reader = new FileReader()
  reader.onloadend = () => {
    setSelectedImage(reader.result as string)
    setAnalysis(null)
  }
  reader.readAsDataURL(file)
}
```

Listing 15: Validacion de tipos de archivo

10 Mejores Practicas de Desarrollo

10.1 Seguridad

- **Validacion de Autenticacion**: Verificar usuario en cada request
- **Limite de Tamaño**: Validar tamaño de imagenes subidas
- **Sanitizacion**: Limpiar inputs del usuario
- **API Keys**: Proteger claves de Gemini en variables de entorno
- **Rate Limiting**: Implementar limites de uso para APIs

10.2 Performance

- **Lazy Loading**: Cargar componentes bajo demanda
- **Optimizacion de Imagenes**: Comprimir imagenes antes del envio
- **Cache de Respuestas**: Almacenar respuestas frecuentes
- **Debouncing**: Evitar requests excesivos en chat
- **Loading States**: Feedback visual durante procesamiento

10.3 Mantenibilidad

- **TypeScript**: Tipado estricto en todos los componentes
- **Interfaces**: Definir interfaces claras para props
- **Error Handling**: Manejo robusto de errores
- **Logging**: Registro detallado para debugging
- **Testing**: Tests unitarios para funciones criticas

11 Solucion de Problemas

11.1 Problemas Comunes

11.1.1 Error: API key no configurada

Sintomas: Error .APIkeynoconfiguradaenrespuestas **Causa**: Variables de entorno no configuradas **Solucion**:

1. Crear archivo `.env.local`
2. Agregar $\texttt{GOOGLE}_G ENERATIVE_A I_A PI_K EY = tu\_clave$
2. `Reiniciar servidor de desarrollo`

11.1.2 Error: Imagen no analizable

Sintomas: Analisis falla o retorna error **Causa**: Imagen corrupta, muy grande o formato no soportado **Solucion**:

1. Verificar formato de imagen (JPG, PNG, WEBP)
2. Reducir tamaño de imagen
3. Asegurar buena iluminacion en fotos

11.1.3 Error: Chat no responde

Sintomas: Mensajes no se procesan o timeout **Causa**: Problemas de red o limite de cuota de API **Solucion**:

1. Verificar conexion a internet
2. Comprobar cuota de Gemini API
3. Revisar logs del servidor

11.2 Debugging

11.2.1 Logs del Cliente

```
// Habilitar logs detallados
console.log('User Profile:', userProfile)
console.log('Messages:', messages)
console.log('Selected Image:', selectedImage)

// Verificar estado de carga
console.log('Is Loading:', isLoading)
console.log('Is Analyzing:', isAnalyzing)
```

Listing 16: Debugging en cliente

11.2.2 Logs del Servidor

```
// En API routes
console.log('Request body:', await request.json())
console.log('User authenticated:', !!user)
console.log('Image size:', image.length)

// En analisis de IA
console.log('Analysis result:', text)
console.log('Confidence level:', object.confidence)
```

Listing 17: Debugging en servidor

12 Conclusion

El modulo de Asistente Fitness representa la vanguardia en aplicaciones de salud y fitness, combinando inteligencia artificial avanzada con personalizacion profunda. Su arquitectura modular y su integracion completa con el ecosistema FitTrack lo convierten en una herramienta poderosa para usuarios que buscan optimizar su nutricion y rendimiento.

Caracteristicas destacadas:

- Integracion completa con datos de usuario
- IA de ultima generacion para analisis preciso
- Interfaz intuitiva y responsiva
- Personalizacion basada en objetivos reales
- Calculos nutricionales cientificamente validados
- Arquitectura escalable y mantenible

Para contribuir al desarrollo del modulo:

1. Seguir las convenciones de TypeScript establecidas
2. Implementar tests para nuevas funcionalidades
3. Documentar cambios en prompts de IA
4. Mantener compatibilidad con APIs de Gemini
5. Respetar limites de cuota de APIs externas