Manual de Programador

FitTrack

Aplicación Integral de Seguimiento Fitness

Integrantes:

Santiago Ibarra

Carlos Insaurralde

Santino Gómez García

Curso: 7°2

Este manual proporciona una guía completa para desarrolladores que trabajen en el proyecto FitTrack, incluyendo arquitectura, configuración, estructura de base de datos y mejores prácticas de desarrollo.

Versión 1.0

15 de octubre de 2025

Índice

1 Introducción

FitTrack es una aplicación web integral de seguimiento fitness desarrollada con Next.js, React y Supabase. Esta sección introduce el propósito del manual, la audiencia a la que está dirigido y las características clave de la aplicación.

1.1 Propósito del Manual

El objetivo de este documento es servir como una guía técnica centralizada para los desarrolladores que trabajan en el proyecto FitTrack. Proporciona información detallada sobre la arquitectura del sistema, las tecnologías utilizadas, la estructura del código, las convenciones de codificación y los procedimientos de despliegue.

1.2 Audiencia Objetivo

Este manual está dirigido a desarrolladores de software, arquitectos de sistemas y cualquier miembro del equipo técnico que necesite comprender el funcionamiento interno de la aplicación FitTrack para su desarrollo, mantenimiento o expansión.

1.3 Características Principales

- Sistema de autenticación completo con roles (Usuario, Profesional, Administrador).
- Registro de entrenamientos de gimnasio con ejercicios personalizados.
- Seguimiento de sesiones de running con métricas detalladas.
- Análisis nutricional de comidas mediante Google Gemini AI.
- Sistema de mensajería entre usuarios y profesionales.
- Monitoreo de métricas de salud (IMC, presión arterial, etc.).
- Panel de administración y configuraciones de accesibilidad avanzadas.

2 Arquitectura y Stack Tecnológico

Esta sección detalla la arquitectura de la aplicación y las tecnologías fundamentales sobre las que se construye FitTrack.

2.1 Tecnologías Principales

Componente	Tecnología
Frontend Framework	Next.js 15.2.4
UI Library	React 19
Styling	Tailwind CSS 4.1.9
Database	Supabase (PostgreSQL)
Authentication	Supabase Auth
AI Integration	Google Gemini 1.5 Flash
UI Components	Radix UI + shadcn/ui
State Management	React Hooks + Context
Type Safety	TypeScript 5

Cuadro 1: Stack Tecnológico Principal

2.2 Dependencias Clave

```
{
  "dependencies": {
    "next": "15.2.4",
    "react": "^19",
    "react-dom": "^19",
    "@supabase/supabase-js": "latest",
    "@supabase/ssr": "latest",
    "@ai-sdk/google": "latest",
    "@google/generative-ai": "latest",
    "ai": "latest",
    "tailwindcss": "^4.1.9",
    "typescript": "^5"
  }
}
```

Listing 1: package.json - Dependencias principales

2.3 Arquitectura de la Aplicación

La aplicación sigue una arquitectura de capas con una clara separación de responsabilidades para garantizar la modularidad y escalabilidad del sistema:

1. **Capa de Presentación (Frontend)**: Construida con componentes de React y estilizada con Tailwind CSS. Se encarga de toda la interacción con el usuario.

2. **Capa de Lógica de Negocio (Backend)**: Implementada mediante Server Actions y API Routes de Next.js. Contiene la lógica principal de la aplicación.

3. **Capa de Datos (Persistencia)**: Gestionada por Supabase, utilizando una base de datos PostgreSQL con Row Level Security (RLS) para el control de acceso a los datos.

4. **Capa de Servicios Externos**: Integra APIs de terceros, como Google Gemini para el análisis nutricional, desacoplando estas funcionalidades del núcleo de la aplicación.

3 Estructura del Proyecto

La organización del código fuente es fundamental para la mantenibilidad. A continuación se describe la estructura de directorios principal del proyecto.

3.1 Organización de Directorios

```
FitTrack/
+-- app/                    # App Router: contiene todas las rutas y paginas.
|   +-- api/                # Rutas de API para funcionalidades especificas.
|   +-- auth/               # Flujos de autenticacion (login, registro, etc.).
|   +-- (modules)/          # Directorios para cada modulo principal (gym, running, etc.).
|   +-- admin/              # Panel de administracion.
+-- components/             # Componentes de React reutilizables.
|   +-- ui/                 # Componentes base de UI (shadcn/ui).
|   +-- shared/             # Componentes compartidos especificos de la aplicacion.
+-- lib/                    # Funciones de utilidad, hooks y Server Actions.
|   +-- supabase/           # Configuracion del cliente de Supabase.
|   +-- *-actions.ts        # Server Actions por modulo.
+-- scripts/                # Scripts de base de datos (migraciones, seeds).
+-- public/                 # Archivos estaticos (imagenes, fuentes, etc.).
```

Listing 2: Estructura de directorios principal

4 Base de Datos

La persistencia de datos de FitTrack se gestiona con Supabase, que utiliza una base de datos PostgreSQL. Esta sección describe el esquema, las políticas de seguridad y los scripts de inicialización.

4.1 Esquema General y Tablas Principales

El esquema está organizado en tres categorías lógicas de tablas:

4.1.1 Tablas de Usuario

Gestionan la información del perfil, roles y preferencias del usuario.

- `auth.users`: Almacena la información de autenticación (gestionado por Supabase Auth).
- `user_profiles`: Contiene datos adicionales del perfil del usuario.
- `user_roles`: Asigna roles (usuario, profesional, admin) para control de acceso.
- `user_preferences`: Guarda configuraciones de accesibilidad y otras preferencias.

4.1.2 Tablas de Actividad

Registran las acciones y datos generados por los usuarios.

- `gym_workouts`: Almacena cada ejercicio individual realizado en el gimnasio.
- `routines`: Guarda las rutinas de entrenamiento personalizadas creadas por los usuarios.
- `routine_exercises`: Tabla de unión que define los ejercicios dentro de una rutina.
- `running_sessions`: Registra las sesiones de carrera.
- `meals`: Contiene la información de las comidas registradas.

4.1.3 Tablas de Sistema y Catálogos

Contienen datos de soporte y catálogos administrables.

- `gym_exercises`: Catálogo de ejercicios de gimnasio predefinidos y administrables.
- `health_metrics`: Almacena métricas de salud como IMC, presión arterial, etc.
- `conversations` y `messages`: Soportan el sistema de mensajería interna.

4.2 Políticas de Seguridad (Row Level Security - RLS)

Para garantizar la privacidad y seguridad de los datos, todas las tablas sensibles implementan RLS. Estas políticas aseguran que los usuarios solo puedan acceder y modificar su propia información. A continuación, un ejemplo representativo:

```
-- Habilitar RLS en la tabla
ALTER TABLE public.gym_workouts ENABLE ROW LEVEL SECURITY;

-- Los usuarios solo pueden ver sus propios entrenamientos.
CREATE POLICY "Permitir acceso de lectura a los propios datos"
ON public.gym_workouts FOR SELECT
USING (auth.uid() = user_id);

-- Los usuarios solo pueden insertar entrenamientos para si mismos.
CREATE POLICY "Permitir insercion de los propios datos"
ON public.gym_workouts FOR INSERT
WITH CHECK (auth.uid() = user_id);

-- Los usuarios solo pueden actualizar o eliminar sus propios registros.
CREATE POLICY "Permitir modificacion de los propios datos"
ON public.gym_workouts FOR UPDATE USING (auth.uid() = user_id);

CREATE POLICY "Permitir eliminacion de los propios datos"
ON public.gym_workouts FOR DELETE USING (auth.uid() = user_id);
```

Listing 3: Ejemplo de políticas RLS para la tabla `gym_workouts`

4.3 Scripts de Inicialización

El directorio `/scripts` contiene los archivos SQL necesarios para inicializar y migrar el esquema de la base de datos. Deben ejecutarse en el orden numérico indicado por su prefijo para asegurar que las dependencias (como claves foráneas) se resuelvan correctamente.

5 Módulos de la Aplicación

La lógica de la aplicación está organizada en módulos funcionales. Esta sección describe la arquitectura y los componentes clave de cada módulo, comenzando por el de Gimnasio.

6 Módulo de Gimnasio

El módulo de gimnasio es un componente central de FitTrack. Permite a los usuarios registrar entrenamientos, crear y gestionar rutinas, y seguir su progreso a lo largo del tiempo.

6.1 Características del Módulo de Gimnasio

- **Registro de Entrenamientos**: Interfaz para registrar ejercicios individuales con peso, repeticiones y series.
- **Gestión de Rutinas**: Funcionalidad para crear, editar, eliminar y ejecutar rutinas personalizadas.
- **Catálogo de Ejercicios**: Acceso a una base de datos de ejercicios predefinidos, que es administrada por los administradores del sistema.
- **Historial y Progreso**: Visualización del historial de entrenamientos y seguimiento del progreso a lo largo del tiempo.

6.2 Server Actions del Módulo de Gimnasio

Toda la lógica de negocio se maneja a través de Server Actions en el archivo `lib/gym-actions.ts`. Estas funciones se ejecutan en el servidor y son responsables de interactuar con la base de datos de forma segura.

```
1  "use server"
2
3  import { revalidatePath } from "next/cache"
4  import { createClient } from "@/lib/supabase/server"
5
6  export async function createWorkout(prevState: any, formData: FormData)
       {
7    // 1. Extraer y validar datos del formulario
8    const exercise_name = formData.get("exercise_name")?.toString();
9    if (!exercise_name) {
10     return { error: "El nombre del ejercicio es requerido." };
11   }
12
13   // 2. Verificar la autenticacion del usuario
14   const supabase = await createClient();
15   const { data: { user } } = await supabase.auth.getUser();
16   if (!user) {
17     return { error: "Accion no autorizada." };
18   }
19
20   // 3. Preparar y sanitizar los datos para la insercion
21   const workoutData = {
```

```
    user_id: user.id,
    exercise_name,
    weight_kg: Number(formData.get("weight_kg")) || null,
    repetitions: Number(formData.get("repetitions")) || null,
    sets: Number(formData.get("sets")) || null,
  };

  // 4. Insertar en la base de datos y manejar errores
  const { error } = await supabase.from("gym_workouts").insert(
   workoutData);
  if (error) {
    console.error("Error en createWorkout:", error);
    return { error: "No se pudo guardar el ejercicio." };
  }

  // 5. Revalidar el cache para actualizar la UI
  revalidatePath("/gym");
  return { success: true };
}
```

Listing 4: Función ‘createWorkout‘ en ‘gym-actions.ts‘

Flujo de la Acción:

- **Validación de Entrada**: Se asegura de que los datos requeridos, como el nombre del ejercicio, estén presentes.
- **Autenticación**: Confirma que la solicitud proviene de un usuario autenticado antes de interactuar con la base de datos.
- **Sanitización de Datos**: Convierte los datos del formulario a los tipos correctos (e.g., ‘Number‘) y maneja valores nulos.
- **Interacción con la Base de Datos**: Realiza la operación de inserción en la tabla `gym_workouts`.
- **Revalidación de Caché**: Llama a `revalidatePath` de Next.js para que la interfaz de usuario refleje los nuevos datos sin necesidad de recargar la página.

6.2.1 Componentes Principales de React

```
"use server"

import { revalidatePath } from "next/cache"
import { createClient } from "@/lib/supabase/server"

export async function createWorkout(prevState: any, formData: FormData)
   {
  // Extraer datos del formulario
  const exercise\_name = formData.get("exercise\_name")?.toString()
  const weight\_kg = formData.get("weight\_kg")?.toString()
  const repetitions = formData.get("repetitions")?.toString()
  const sets = formData.get("sets")?.toString()
  const image\_url = formData.get("image\_url")?.toString()

  // Validacion basica
  if (!exercise\_name) {
```

```
    return { error: "El nombre del ejercicio es requerido" }
  }

  // Verificar autenticacion
  const supabase = await createClient()
  const {
    data: { user },
  } = await supabase.auth.getUser()

  if (!user) {
    return { error: "Usuario no autenticado" }
  }

  try {
    // Preparar datos para insercion
    const insertData = {
      user\_id: user.id,
      exercise\_name,
      weight\_kg: weight\_kg && weight\_kg.trim() !== "" ?
        Math.max(0, Number.parseFloat(weight\_kg)) : null,
      repetitions: repetitions && repetitions.trim() !== "" ?
        Math.max(1, Number.parseInt(repetitions)) : null,
      sets: sets && sets.trim() !== "" ?
        Math.max(1, Number.parseInt(sets)) : null,
      image\_url: image\_url && image\_url.trim() !== "" ?
        image\_url.trim() : null,
    }

    // Insertar en base de datos
    const { error } = await supabase
      .from("gym\_workouts")
      .insert(insertData)

    if (error) {
      console.error("Database error:", error)
      return { error: "Error al guardar el ejercicio" }
    }

    // Revalidar cache
    revalidatePath("/gym")
    return { success: true }
  } catch (error) {
    console.error("Error:", error)
    return { error: "Error al guardar el ejercicio" }
  }
}
```

Listing 5: Funcion createWorkout completa

Caracteristicas importantes:

- **Validacion de entrada**: Verifica que el nombre del ejercicio este presente
- **Autenticacion**: Confirma que el usuario este autenticado
- **Sanitizacion**: Convierte y valida valores numericos
- **Manejo de errores**: Captura y reporta errores de base de datos
- **Revalidacion**: Actualiza el cache de Next.js

6.3 Componentes del Módulo de Gimnasio

6.3.1 Página Principal - GymPage

El componente principal que coordina toda la funcionalidad del modulo de gimnasio.

```
"use client"

import { useState } from "react"
import { Dumbbell, ArrowLeft } from "lucide-react"
import Link from "next/link"
import { Button } from "@/components/ui/button"
import WorkoutForm from "@/components/gym/workout-form"
import WorkoutList from "@/components/gym/workout-list"
import RoutineList from "@/components/gym/routine-list"
import RoutineForm from "@/components/gym/routine-form"
import RoutineDetail from "@/components/gym/routine-detail"
import ExerciseHistory from "@/components/gym/exercise-history"
import GymMetrics from "@/components/gym/gym-metrics"

// Interfaces TypeScript
interface Workout {
  id: string
  exercise\_name: string
  weight\_kg: number | null
  repetitions: number | null
  sets: number | null
  created\_at: string
}

type ViewMode = "routines" | "individual" | "routine-detail" | "create-
    routine" | "history" | "metrics"

export default function GymPage() {
  // Estados del componente
  const [refreshTrigger, setRefreshTrigger] = useState(0)
  const [editingWorkout, setEditingWorkout] = useState<Workout | null>(
    null)
  const [viewMode, setViewMode] = useState<ViewMode>("routines")
  const [selectedRoutine, setSelectedRoutine] = useState<{ id: string;
    name: string } | null>(null)

  // Handlers para diferentes acciones
  const handleWorkoutAdded = () => {
    setRefreshTrigger((prev) => prev + 1)
  }

  const handleEditWorkout = (workout: Workout) => {
    setEditingWorkout(workout)
  }

  const handleEditComplete = () => {
    setEditingWorkout(null)
    setRefreshTrigger((prev) => prev + 1)
  }

  const handleViewRoutine = (routineId: string, routineName: string) =>
    {
    setSelectedRoutine({ id: routineId, name: routineName })
```

```
50      setViewMode("routine-detail")
51    }
52
53    const handleCreateRoutine = () => {
54      setViewMode("create-routine")
55    }
56
57    const handleRoutineCreated = () => {
58      setViewMode("routines")
59      setRefreshTrigger((prev) => prev + 1)
60    }
61
62    const handleBackToRoutines = () => {
63      setSelectedRoutine(null)
64      setViewMode("routines")
65      setRefreshTrigger((prev) => prev + 1)
66    }
67
68    return (
69      <div className="min-h-screen bg-gradient-to-br from-blue-50 to-
       indigo-100 dark:from-gray-900 dark:to-gray-800">
70        <div className="container mx-auto px-4 py-8 max-w-4xl">
71          {/* Header con navegacion */}
72          <div className="mb-8">
73            <div className="flex items-center gap-4 mb-4">
74              <Button variant="outline" size="sm" asChild>
75                <Link href="/">
76                  <ArrowLeft className="h-4 w-4 mr-2" />
77                  Volver
78                </Link>
79              </Button>
80            </div>
81            <div className="flex items-center gap-3 mb-2">
82              <Dumbbell className="h-8 w-8 text-blue-600 dark:text-blue
       -400" />
83              <h1 className="text-3xl font-bold text-gray-900 dark:text-
       white">Gimnasio</h1>
84            </div>
85            <p className="text-gray-600 dark:text-gray-300">
86              Organiza tus entrenamientos por rutinas o registra
       ejercicios individuales
87            </p>
88          </div>
89
90          {/* Pestanas de navegacion */}
91          <div className="flex gap-2 mb-6 flex-wrap">
92            <button
93              onClick={() => setViewMode("routines")}
94              className={`px-4 py-2 rounded-lg font-medium transition-
       colors ${
95                viewMode === "routines" || viewMode === "routine-detail"
       || viewMode === "create-routine"
96                  ? "bg-blue-600 text-white"
97                  : "bg-gray-100 dark:bg-gray-700 text-gray-700 dark:text-
       gray-300 hover:bg-gray-200 dark:hover:bg-gray-600"
98              }`}
99            >
100             Rutinas
```

```
101           </button>
102           <button
103             onClick={() => setViewMode("individual")}
104             className={`px-4 py-2 rounded-lg font-medium transition-
    colors ${
105               viewMode === "individual" ? "bg-blue-600 text-white" : "bg
    -gray-100 dark:bg-gray-700 text-gray-700 dark:text-gray-300 hover:bg-
    gray-200 dark:hover:bg-gray-600"
106             }`}
107           >
108             Ejercicios Individuales
109           </button>
110           <button
111             onClick={() => setViewMode("history")}
112             className={`px-4 py-2 rounded-lg font-medium transition-
    colors ${
113               viewMode === "history" ? "bg-blue-600 text-white" : "bg-
    gray-100 dark:bg-gray-700 text-gray-700 dark:text-gray-300 hover:bg-
    gray-200 dark:hover:bg-gray-600"
114             }`}
115           >
116             Historial
117           </button>
118           <button
119             onClick={() => setViewMode("metrics")}
120             className={`px-4 py-2 rounded-lg font-medium transition-
    colors ${
121               viewMode === "metrics" ? "bg-blue-600 text-white" : "bg-
    gray-100 dark:bg-gray-700 text-gray-700 dark:text-gray-300 hover:bg-
    gray-200 dark:hover:bg-gray-600"
122             }`}
123           >
124             Metricas
125           </button>
126         </div>
127
128         {/* Renderizado condicional de componentes */}
129         <div className="grid gap-6">
130           {viewMode === "routines" && (
131             <RoutineList
132               refreshTrigger={refreshTrigger}
133               onViewRoutine={handleViewRoutine}
134               onCreateRoutine={handleCreateRoutine}
135             />
136           )}
137
138           {viewMode === "create-routine" && (
139             <RoutineForm
140               onRoutineCreated={handleRoutineCreated}
141               onCancel={() => setViewMode("routines")}
142             />
143           )}
144
145           {viewMode === "routine-detail" && selectedRoutine && (
146             <RoutineDetail
147               routineId={selectedRoutine.id}
148               routineName={selectedRoutine.name}
149               onBack={handleBackToRoutines}
```

```
            />
          )}

          {viewMode === "individual" && (
            <>
              <WorkoutForm
                onWorkoutAdded={handleWorkoutAdded}
                editWorkout={editingWorkout}
                onEditComplete={handleEditComplete}
              />
              <WorkoutList
                refreshTrigger={refreshTrigger}
                onEditWorkout={handleEditWorkout}
              />
            </>
          )}

          {viewMode === "history" && <ExerciseHistory />}
          {viewMode === "metrics" && <GymMetrics />}
        </div>
      </div>
    </div>
  )
}
```

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

Caracteristicas del componente:

- **Estado centralizado**: Maneja todos los estados de la aplicacion
- **Navegacion por pestanas**: Interfaz intuitiva para cambiar entre vistas
- **Renderizado condicional**: Muestra diferentes componentes segun el modo
- **Handlers de eventos**: Gestiona todas las interacciones del usuario
- **TypeScript**: Tipado estricto para mayor seguridad

6.4 Ejemplos del Módulo de Gimnasio

6.4.1 Ejemplos de Inserción a Base de Datos

6.4.2 Crear una Rutina Completa

```
-- 1. Crear la rutina
INSERT INTO public.routines (
    user_id,
    name,
    description
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000',
    'Rutina de Pecho y Triceps',
    'Rutina enfocada en el desarrollo del pecho y triceps,
   ideal para principiantes'
);
```

```
11
12 -- Obtener el ID de la rutina recien creada
13 -- (En la aplicacion esto se maneja automaticamente)
14
15 -- 2. Agregar ejercicios a la rutina
16 INSERT INTO public.routine_exercises (
17     routine_id,
18     exercise_name,
19     weight,
20     repetitions,
21     sets,
22     order_index
23 ) VALUES
24     ('456e7890-e89b-12d3-a456-426614174001', 'Press de Banca
   ', 80.00, 12, 3, 0),
25     ('456e7890-e89b-12d3-a456-426614174001', 'Aperturas con
   Mancuernas', 25.00, 15, 3, 1),
26     ('456e7890-e89b-12d3-a456-426614174001', 'Press Frances',
    30.00, 12, 3, 2),
27     ('456e7890-e89b-12d3-a456-426614174001', 'Fondos para
   Triceps', 0.00, 10, 3, 3);
```

Listing 7: Ejemplo de creacion de rutina completa

6.4.3 Consultas Utiles

```
1 -- Obtener todos los entrenamientos de un usuario con detalles
2 SELECT
3     gw.exercise_name,
4     gw.weight_kg,
5     gw.repetitions,
6     gw.sets,
7     gw.created_at,
8     (gw.weight_kg * gw.repetitions * gw.sets) as volumen_total
9 FROM public.gym_workouts gw
10 WHERE gw.user_id = '123e4567-e89b-12d3-a456-426614174000'
11 ORDER BY gw.created_at DESC;
12
13 -- Obtener progreso de un ejercicio especifico
14 SELECT
15     exercise_name,
16     weight_kg,
17     repetitions,
18     sets,
19     created_at,
20     (weight_kg * repetitions * sets) as volumen
21 FROM public.gym_workouts
22 WHERE user_id = '123e4567-e89b-12d3-a456-426614174000'
23   AND exercise_name = 'Press de Banca'
24 ORDER BY created_at ASC;
25
26 -- Obtener rutinas con conteo de ejercicios
27 SELECT
```

```
    r.name,
    r.description,
    r.created_at,
    COUNT(re.id) as total_ejercicios
FROM public.routines r
LEFT JOIN public.routine_exercises re ON r.id = re.routine_id
WHERE r.user_id = '123e4567-e89b-12d3-a456-426614174000'
GROUP BY r.id, r.name, r.description, r.created_at
ORDER BY r.created_at DESC;

-- Obtener ejercicios de una rutina especifica en orden
SELECT
    re.exercise_name,
    re.weight,
    re.repetitions,
    re.sets,
    re.order_index
FROM public.routine_exercises re
WHERE re.routine_id = '456e7890-e89b-12d3-a456-426614174001'
ORDER BY re.order_index ASC;
```

Listing 8: Consultas utiles para analisis

6.5 Flujos de Datos Completos

6.5.1 Flujo de Creacion de Entrenamiento

1. **Usuario completa formulario** en `WorkoutForm`
2. **Validacion en cliente** de campos requeridos
3. **Envio a Server Action** `createWorkout`
4. **Verificacion de autenticacion** en servidor
5. **Sanitizacion de datos** (conversion de tipos)
6. **Insercion en base de datos** tabla `gym_workouts`
7. **Revalidacion de cache** con `revalidatePath`
8. **Actualizacion de UI** con nuevo entrenamiento

6.5.2 Flujo de Gestion de Rutinas

1. **Creacion de rutina** con `createRoutine`
2. **Insercion en tabla** `routines`
3. **Agregar ejercicios** con `addExerciseToRoutine`
4. **Insercion en tabla** `routine_exercises` con `order_index`
5. **Visualizacion de rutina** con `getRoutineExercises`
6. **Ejecucion de rutina** (registro de entrenamientos individuales)

7 Caracteristicas Avanzadas

7.1 Selector de Ejercicios

El modulo incluye un selector modal avanzado que permite:

- **Catalogo de ejercicios**: Acceso a ejercicios administrados
- **Filtrado por categoria**: Pecho, Biceps, Triceps, etc.
- **Busqueda de ejercicios**: Filtrado por nombre
- **Ejercicios personalizados**: Creacion de ejercicios unicos
- **Imagenes de ejercicios**: Visualizacion de tecnicas

7.2 Metricas y Estadisticas

El componente `GymMetrics` proporciona:

- **Graficos de progreso**: Evolucion del peso a lo largo del tiempo
- **Estadisticas de volumen**: Calculo de volumen total (peso $\times$ reps $\times$ series)
- **Tendencias de entrenamiento**: Frecuencia y consistencia
- **Comparativas temporales**: Progreso mensual/semanal
- **Analisis por ejercicio**: Progreso especifico por ejercicio

7.3 Historial de Ejercicios

El componente `ExerciseHistory` permite:

- **Historial completo**: Todos los entrenamientos registrados
- **Filtrado por ejercicio**: Ver progreso de un ejercicio especifico
- **Analisis temporal**: Progreso a lo largo del tiempo
- **Exportacion de datos**: Para analisis externo
- **Busqueda avanzada**: Filtros por fecha, ejercicio, peso

8 Mejores Practicas de Desarrollo

8.1 Seguridad

- **Validacion de entrada**: Siempre validar datos en Server Actions
- **Autenticacion**: Verificar usuario en cada operacion
- **RLS**: Usar Row Level Security en todas las tablas
- **Sanitizacion**: Limpiar y convertir datos de entrada
- **Logs de seguridad**: Registrar operaciones sensibles

8.2 Performance

- **Indices de base de datos**: Optimizar consultas frecuentes
- **Paginacion**: Implementar para listas largas
- **Cache**: Usar revalidatePath para actualizar cache
- **Lazy loading**: Cargar componentes bajo demanda
- **Optimizacion de imagenes**: Comprimir y optimizar imagenes

8.3 Mantenibilidad

- **TypeScript**: Usar tipado estricto en todos los componentes
- **Interfaces**: Definir interfaces claras para props
- **Separacion de responsabilidades**: Logica en Server Actions
- **Reutilizacion**: Componentes modulares y reutilizables
- **Documentacion**: Comentar codigo complejo

9 Solucion de Problemas

9.1 Problemas Comunes

9.1.1 Error: Tabla no existe

Sintomas: Error al intentar insertar o consultar datos **Causa**: Las tablas no han sido creadas en la base de datos **Solucion**: Ejecutar los scripts SQL en orden:

1. `01-create-database-schema.sql`
2. `01-create-user-schema.sql`
3. `08-verify-routines-tables.sql`
4. `26-create-gym-exercises-table.sql`

9.1.2 Error: Usuario no autenticado

Sintomas: Server Actions retornan error de autenticacion **Causa**: Usuario no esta logueado o sesion expirada **Solucion**: Verificar estado de autenticacion y redirigir a login

9.1.3 Error: Politica RLS violada

Sintomas: Error al acceder a datos de otros usuarios **Causa**: Politicas RLS mal configuradas **Solucion**: Verificar y corregir politicas RLS

9.2 Debugging

9.2.1 Logs del Cliente

```
// Habilitar logs detallados
localStorage.setItem('debug', 'true');

// Verificar estado de autenticacion
console.log('User:', user);
console.log('Session:', session);

// Verificar datos de entrenamientos
console.log('Workouts:', workouts);
```

Listing 9: Debugging en cliente

9.2.2 Logs del Servidor

```
// En Server Actions
console.log('Action called with:', { userId, data });

// En consultas de base de datos
console.log('Query result:', { data, error });

// En validaciones
console.log('Validation result:', validationResult);
```

Listing 10: Debugging en servidor

10 Conclusion

El modulo de gimnasio de FitTrack es un sistema completo y robusto que permite a los usuarios gestionar sus entrenamientos de manera eficiente. Con su arquitectura bien definida, base de datos optimizada y componentes React modernos, proporciona una experiencia de usuario excepcional.

Caracteristicas destacadas:

- Arquitectura escalable y mantenible
- Seguridad robusta con RLS
- Interfaz de usuario intuitiva
- Funcionalidades avanzadas de analisis
- Codigo bien documentado y tipado

Para contribuir al desarrollo del modulo:

1. Seguir las convenciones establecidas
2. Implementar tests apropiados
3. Documentar cambios significativos

4. Mantener la compatibilidad con la base de datos
5. Respetar las politicas de seguridad

11 Módulo de Running

El módulo de running es un componente fundamental de FitTrack que permite a los usuarios registrar, analizar y hacer seguimiento de sus sesiones de carrera. Proporciona herramientas completas para el monitoreo del progreso, cálculo automático de métricas y visualización de estadísticas detalladas.

11.1 Características del Módulo de Running

- **Registro de Sesiones**: Permite registrar sesiones de carrera con duracion, distancia y ritmo
- **Calculo Automatico de Ritmo**: Calcula automaticamente el ritmo por kilometro basado en duracion y distancia
- **Estadisticas Detalladas**: Proporciona metricas completas como total de sesiones, distancia acumulada, tiempo total, ritmo promedio y mejor ritmo
- **Historial de Sesiones**: Mantiene un registro completo de todas las sesiones de carrera
- **Graficos de Progreso**: Visualizacion de tendencias y progreso a lo largo del tiempo
- **Analisis de Rendimiento**: Comparacion de sesiones y identificacion de mejoras
- **Interfaz Intuitiva**: Diseño limpio y facil de usar para registro rapido

11.2 Arquitectura General

El modulo sigue una arquitectura de capas bien definida:

1. **Capa de Presentacion**: Componentes React con TypeScript
2. **Capa de Logica**: Server Actions de Next.js
3. **Capa de Datos**: Supabase PostgreSQL con RLS
4. **Capa de Servicios**: Calculos y utilidades de metricas

11.3 Base de Datos del Módulo de Running

11.3.1 Diagrama de Relaciones

11.4 Tabla running_sessions

Esta tabla almacena las sesiones de carrera realizadas por los usuarios.

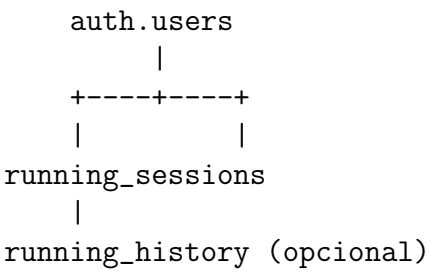

Figura 1: Diagrama de relaciones de las tablas del modulo de running

```sql
CREATE TABLE IF NOT EXISTS public.running\_sessions (
    id UUID DEFAULT gen\_random\_uuid() PRIMARY KEY,
    user\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
    duration\_minutes INTEGER NOT NULL CHECK (duration\_minutes > 0),
    distance\_km NUMERIC(5,2) NOT NULL CHECK (distance\_km > 0),
    pace\_min\_km NUMERIC(5,2),
    created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indices para optimizacion
CREATE INDEX IF NOT EXISTS idx\_running\_sessions\_user\_id ON public.
   running\_sessions(user\_id);
CREATE INDEX IF NOT EXISTS idx\_running\_sessions\_created\_at ON public
   .running\_sessions(created\_at);
CREATE INDEX IF NOT EXISTS idx\_running\_sessions\_distance ON public.
   running\_sessions(distance\_km);
CREATE INDEX IF NOT EXISTS idx\_running\_sessions\_pace ON public.
   running\_sessions(pace\_min\_km);
```

Listing 11: Estructura completa de `running_sessions`

Descripcion de campos:

- `id`: Identificador unico UUID generado automaticamente
- `user_id`: Referencia al usuario propietario de la sesion
- `duration_minutes`: Duracion de la sesion en minutos (requerido, ¿ 0)
- `distance_km`: Distancia recorrida en kilometros (requerido, ¿ 0)
- `pace_min_km`: Ritmo en minutos por kilometro (opcional, calculado automaticamente)
- `created_at`: Timestamp de creacion automatico

11.4.1 Lógica de Negocio (Server Actions)

Las operaciones de backend para el módulo de running se encuentran en `lib/running-actions.ts`. A continuación se describen las acciones principales.

```
"use server"

// createRunningSession: Guarda una nueva sesion de carrera.

```

```
}xport async function createRunningSession(prevState: any, formData:
    FormData) {
  // 1. Validar duracion y distancia.
  // 2. Autenticar al usuario.
  // 3. Calcular el ritmo si es necesario.
  // 4. Insertar en la tabla `running_sessions`.
  // 5. Revalidar el path `/running`.
}

// getRunningSessions: Obtiene todas las sesiones de un usuario.
export async function getRunningSessions() {
  // 1. Autenticar al usuario.
  // 2. Consultar la tabla `running_sessions`.
  // 3. Ordenar por fecha de creacion descendente.
  // 4. Devolver los datos o un array vacio.
}

// deleteRunningSession: Elimina una sesion de carrera.
export async function deleteRunningSession(sessionId: string) {
  // 1. Autenticar al usuario.
  // 2. Eliminar el registro de la tabla `running_sessions` donde el ID
    y el user_id coincidan.
  // 3. Revalidar el path `/running`.
}
```

Listing 12: Funciones principales en `running-actions.ts`

11.4.2 Componentes Principales de React

La interfaz de usuario del módulo se compone de varios componentes que trabajan en conjunto.

- `app/running/page.tsx`: Es el componente principal que organiza el layout de la página. Gestiona la navegación entre la vista de "Sesiones" y "Gráficos" y coordina la actualización de datos entre sus componentes hijos.
- `components/running/running-form.tsx`: Un formulario que utiliza `useActionState` para invocar la Server Action `createRunningSession`. Muestra un botón para abrir el formulario y gestiona el estado de envío y los errores de validación.
- `components/running/running-list.tsx`: Muestra una lista de todas las sesiones de carrera del usuario. Obtiene los datos llamando a la Server Action `getRunningSessions` y permite eliminar sesiones individuales.
- `components/running/running-stats.tsx`: Calcula y muestra estadísticas agregadas, como la distancia total, el ritmo promedio y la mejor sesión, basándose en los datos obtenidos.
- `components/running/running-charts.tsx`: Visualiza los datos históricos en forma de gráficos para que el usuario pueda analizar su progreso a lo largo del tiempo.

Caracteristicas del formulario:

- **Estado colapsable**: Se muestra como boton inicialmente, se expande al hacer clic

- **Validacion en cliente**: Campos requeridos y validacion de tipos
- **Calculo automatico**: El ritmo se calcula automaticamente si no se proporciona
- **Manejo de errores**: Muestra errores de validacion y base de datos
- **Feedback visual**: Indicador de carga durante el envio

11.5 Componentes del Módulo de Running

11.5.1 Componente RunningStats

Componente que calcula y muestra estadisticas detalladas de las sesiones de running.

```
interface Stats {
  totalSessions: number
  totalDistance: number
  totalTime: number
  averagePace: number | null
  bestPace: number | null
  longestRun: number
}

export default function RunningStats({ refreshTrigger }: {
   refreshTrigger?: number }) {
  const [stats, setStats] = useState<Stats | null>(null)
  const [loading, setLoading] = useState(true)

  const calculateStats = (sessions: RunningSession[]): Stats => {
    if (sessions.length === 0) {
      return {
        totalSessions: 0,
        totalDistance: 0,
        totalTime: 0,
        averagePace: null,
        bestPace: null,
        longestRun: 0,
      }
    }

    // Calcular metricas basicas
    const totalDistance = sessions.reduce((sum, session) => sum +
   session.distance_km, 0)
    const totalTime = sessions.reduce((sum, session) => sum + session.
   duration_minutes, 0)
    const longestRun = Math.max(...sessions.map((session) => session.
   distance_km))

    // Calcular ritmo promedio y mejor ritmo
    const sessionsWithPace = sessions.filter((session) => session.
   pace_min_km !== null)
    const averagePace =
      sessionsWithPace.length > 0
        ? sessionsWithPace.reduce((sum, session) => sum + (session.
   pace_min_km || 0), 0) / sessionsWithPace.length
        : null

    const bestPace =
```

```
      sessionsWithPace.length > 0
        ? Math.min(...sessionsWithPace.map((session) => session.pace_min_km || Number.POSITIVE_INFINITY))
        : null

    return {
      totalSessions: sessions.length,
      totalDistance,
      totalTime,
      averagePace,
      bestPace,
      longestRun,
    }
  }

  const loadStats = async () => {
    try {
      const sessions = await getRunningSessions()
      const calculatedStats = calculateStats(sessions || [])
      setStats(calculatedStats)
    } catch (error) {
      console.error("Error loading running stats:", error)
    } finally {
      setLoading(false)
    }
  }

  useEffect(() => {
    loadStats()
  }, [refreshTrigger])

  // Funcion para formatear el ritmo
  const formatPace = (pace: number | null) => {
    if (!pace) return "N/A"
    const minutes = Math.floor(pace)
    const seconds = Math.round((pace - minutes) * 60)
    return `${minutes}:${seconds.toString().padStart(2, "0")}`
  }

  // Funcion para formatear el tiempo
  const formatTime = (minutes: number) => {
    const hours = Math.floor(minutes / 60)
    const mins = minutes % 60
    if (hours > 0) {
      return `${hours}h ${mins}m`
    }
    return `${mins}m`
  }

  if (loading) {
    return (
      <Card>
        <CardContent className="p-6">
          <div className="text-center text-gray-500">Cargando estadisticas...</div>
        </CardContent>
      </Card>
    )
```

```
  }

  return (
    <Card>
      <CardHeader>
        <CardTitle>Estadisticas de Running</CardTitle>
        <CardDescription>Resumen de tu actividad de carrera</
    CardDescription>
      </CardHeader>
      <CardContent>
        <div className="grid grid-cols-2 md:grid-cols-3 gap-4">
          <div className="text-center">
            <div className="text-2xl font-bold text-green-600">{stats?.
    totalSessions}</div>
            <div className="text-sm text-gray-600">Sesiones Totales</div
    >
          </div>
          <div className="text-center">
            <div className="text-2xl font-bold text-green-600">{stats?.
    totalDistance.toFixed(1)} km</div>
            <div className="text-sm text-gray-600">Distancia Total</div>
          </div>
          <div className="text-center">
            <div className="text-2xl font-bold text-green-600">{
    formatTime(stats?.totalTime || 0)}</div>
            <div className="text-sm text-gray-600">Tiempo Total</div>
          </div>
          <div className="text-center">
            <div className="text-2xl font-bold text-green-600">{
    formatPace(stats?.averagePace)}</div>
            <div className="text-sm text-gray-600">Ritmo Promedio</div>
          </div>
          <div className="text-center">
            <div className="text-2xl font-bold text-green-600">{
    formatPace(stats?.bestPace)}</div>
            <div className="text-sm text-gray-600">Mejor Ritmo</div>
          </div>
          <div className="text-center">
            <div className="text-2xl font-bold text-green-600">{stats?.
    longestRun.toFixed(1)} km</div>
            <div className="text-sm text-gray-600">Carrera Mas Larga</
    div>
          </div>
        </div>
      </CardContent>
    </Card>
  )
}
```

Listing 13: components/running/running-stats.tsx - Logica de calculo

Metricas calculadas:

- **Sesiones Totales**: Numero total de sesiones registradas
- **Distancia Total**: Suma de todas las distancias recorridas
- **Tiempo Total**: Suma de todos los tiempos de carrera

- **Ritmo Promedio**: Promedio de todos los ritmos registrados
- **Mejor Ritmo**: El ritmo mas rapido registrado
- **Carrera Mas Larga**: La distancia mas larga en una sola sesion

11.5.2 Componente RunningList

Lista que muestra todas las sesiones de running registradas.

```
"use client"

import { useState, useEffect } from "react"
import { Card, CardContent } from "@/components/ui/card"
import { Button } from "@/components/ui/button"
import { Badge } from "@/components/ui/badge"
import { Trash2, Calendar, Clock, MapPin, Zap } from "lucide-react"
import { deleteRunningSession, getRunningSessions } from "@/lib/running-
    actions"
import { format } from "date-fns"
import { es } from "date-fns/locale"

interface RunningSession {
  id: string
  duration_minutes: number
  distance_km: number
  pace_min_km: number | null
  created_at: string
}

export default function RunningList({ refreshTrigger }: { refreshTrigger
    ?: number }) {
  const [sessions, setSessions] = useState<RunningSession[]>([])
  const [loading, setLoading] = useState(true)

  const loadSessions = async () => {
    try {
      const data = await getRunningSessions()
      setSessions(data || [])
    } catch (error) {
      console.error("Error loading running sessions:", error)
    } finally {
      setLoading(false)
    }
  }

  useEffect(() => {
    loadSessions()
  }, [refreshTrigger])

  const handleDelete = async (id: string) => {
    if (confirm("Estas seguro de que quieres eliminar esta sesion?")) {
      const result = await deleteRunningSession(id)
      if (result?.success) {
        loadSessions()
      }
    }
  }
```

```
47
48  const formatPace = (pace: number | null) => {
49    if (!pace) return null
50    const minutes = Math.floor(pace)
51    const seconds = Math.round((pace - minutes) * 60)
52    return `${minutes}:${seconds.toString().padStart(2, "0")}`
53  }
54
55  const formatTime = (minutes: number) => {
56    const hours = Math.floor(minutes / 60)
57    const mins = minutes % 60
58    if (hours > 0) {
59      return `${hours}h ${mins}m`
60    }
61    return `${mins}m`
62  }
63
64  // Estado de carga
65  if (loading) {
66    return (
67      <Card>
68        <CardContent className="p-6">
69          <div className="text-center text-gray-500">Cargando sesiones
     ...</div>
70        </CardContent>
71      </Card>
72    )
73  }
74
75  // Estado vacio
76  if (sessions.length === 0) {
77    return (
78      <Card>
79        <CardContent className="p-6">
80          <div className="text-center text-gray-500">
81            <MapPin className="h-12 w-12 mx-auto mb-4 text-gray-300" />
82            <p>No hay sesiones de running registradas</p>
83            <p className="text-sm">Registra tu primera carrera para
     comenzar</p>
84          </div>
85        </CardContent>
86      </Card>
87    )
88  }
89
90  return (
91    <div className="space-y-4">
92      <h2 className="text-xl font-semibold">Historial de Sesiones</h2>
93      {sessions.map((session) => (
94        <Card key={session.id}>
95          <CardContent className="p-4">
96            <div className="flex items-center justify-between">
97              <div className="flex-1">
98                <div className="flex items-center gap-4 mb-2">
99                  <div className="flex items-center gap-1 text-sm text-
     gray-600">
100                   <Calendar className="h-4 w-4" />
```

```
                    {format(new Date(session.created_at), "dd/MM/yyyy",
    { locale: es })}
                  </div>
                  <div className="flex items-center gap-1 text-sm text-
    gray-600">
                    <Clock className="h-4 w-4" />
                    {formatTime(session.duration_minutes)}
                  </div>
                  <div className="flex items-center gap-1 text-sm text-
    gray-600">
                    <MapPin className="h-4 w-4" />
                    {session.distance_km.toFixed(1)} km
                  </div>
                  {session.pace_min_km && (
                    <div className="flex items-center gap-1 text-sm text
    -gray-600">
                      <Zap className="h-4 w-4" />
                      {formatPace(session.pace_min_km)}/km
                    </div>
                  )}
                </div>

                {/* Badges con metricas */}
                <div className="flex gap-2">
                  <Badge variant="secondary">
                    {session.distance_km.toFixed(1)} km
                  </Badge>
                  <Badge variant="secondary">
                    {formatTime(session.duration_minutes)}
                  </Badge>
                  {session.pace_min_km && (
                    <Badge variant="secondary">
                      {formatPace(session.pace_min_km)}/km
                    </Badge>
                  )}
                </div>
              </div>

              {/* Botones de accion */}
              <div className="flex gap-2">
                <Button
                  variant="outline"
                  size="sm"
                  onClick={() => handleDelete(session.id)}
                  className="text-red-600 hover:text-red-700"
                >
                  <Trash2 className="h-4 w-4" />
                </Button>
              </div>
            </div>
          </CardContent>
        </Card>
      ))}
    </div>
  )
}
```

Listing 14: components/running/running-list.tsx - Estructura principal

Caracteristicas de la lista:

- **Carga asincrona**: Carga las sesiones desde la base de datos
- **Formateo de datos**: Convierte tiempos y ritmos a formato legible
- **Estado vacio**: Muestra mensaje cuando no hay sesiones
- **Eliminacion**: Permite eliminar sesiones con confirmacion
- **Actualizacion automatica**: Se actualiza cuando se agregan nuevas sesiones

11.5.3 Componente RunningCharts

Componente para visualizar graficos de progreso y tendencias.

```
"use client"

import { useState, useEffect } from "react"
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card"
import { getRunningSessions } from "@/lib/running-actions"

interface RunningSession {
  id: string
  duration_minutes: number
  distance_km: number
  pace_min_km: number | null
  created_at: string
}

export default function RunningCharts() {
  const [sessions, setSessions] = useState<RunningSession[]>([])
  const [loading, setLoading] = useState(true)

  useEffect(() => {
    const loadSessions = async () => {
      try {
        const data = await getRunningSessions()
        setSessions(data || [])
      } catch (error) {
        console.error("Error loading running sessions:", error)
      } finally {
        setLoading(false)
      }
    }

    loadSessions()
  }, [])

  if (loading) {
    return (
      <Card>
        <CardContent classNamе="p-6">
          <div classNamе="text-center text-gray-500">Cargando graficos...</div>
        </CardContent>
      </Card>
```

```
    )
  }

  if (sessions.length === 0) {
    return (
      <Card>
        <CardContent className="p-6">
          <div className="text-center text-gray-500">
            <p>No hay datos suficientes para mostrar graficos</p>
            <p className="text-sm">Registra algunas sesiones para ver tu progreso</p>
          </div>
        </CardContent>
      </Card>
    )
  }

  return (
    <div className="space-y-6">
      <Card>
        <CardHeader>
          <CardTitle>Progreso de Distancia</CardTitle>
        </CardHeader>
        <CardContent>
          {/* Aqui se implementarian los graficos con una libreria como Chart.js o Recharts */}
          <div className="h-64 flex items-center justify-center text-gray-500">
            Grafico de progreso de distancia (implementar con libreria de graficos)
          </div>
        </CardContent>
      </Card>

      <Card>
        <CardHeader>
          <CardTitle>Evolucion del Ritmo</CardTitle>
        </CardHeader>
        <CardContent>
          <div className="h-64 flex items-center justify-center text-gray-500">
            Grafico de evolucion del ritmo (implementar con libreria de graficos)
          </div>
        </CardContent>
      </Card>

      <Card>
        <CardHeader>
          <CardTitle>Frecuencia de Entrenamiento</CardTitle>
        </CardHeader>
        <CardContent>
          <div className="h-64 flex items-center justify-center text-gray-500">
            Grafico de frecuencia de entrenamiento (implementar con libreria de graficos)
          </div>
        </CardContent>
```

```
      </Card>
    </div>
  )
}
```

Listing 15: components/running/running-charts.tsx - Estructura basica

Tipos de graficos implementables:

- **Progreso de Distancia**: Linea temporal mostrando la evolucion de las distancias
- **Evolucion del Ritmo**: Grafico de ritmo promedio por sesion
- **Frecuencia de Entrenamiento**: Histograma de sesiones por semana/mes
- **Comparacion de Metricas**: Graficos de barras comparando diferentes periodos

11.6 Ejemplos del Módulo de Running

11.6.1 Ejemplos de Inserción a Base de Datos

11.6.2 Crear una Sesion de Running

```
-- Insertar una sesion de running de 5km en 30 minutos
INSERT INTO public.running_sessions (
    user_id,
    duration_minutes,
    distance_km,
    pace_min_km
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000', -- UUID del usuario
    30,                                     -- 30 minutos
    5.0,                                    -- 5 kilometros
    6.0                                     -- 6 minutos por kilometro
);

-- Insertar una sesion de running de 10km en 50 minutos
INSERT INTO public.running_sessions (
    user_id,
    duration_minutes,
    distance_km,
    pace_min_km
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000',
    50,                                     -- 50 minutos
    10.0,                                   -- 10 kilometros
    5.0                                     -- 5 minutos por kilometro
);

-- Insertar una sesion sin ritmo especificado (se calcula
   automaticamente)
INSERT INTO public.running_sessions (
    user_id,
    duration_minutes,
    distance\_km
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000',
    25,                                     -- 25 minutos
```

```
    4.2                                          -- 4.2 kilometros
    -- pace\_min\_km se calculara como 25/4.2 = 5.95 min/km
);
```

Listing 16: Ejemplo de insercion en `running_sessions`

11.6.3 Consultas Utiles para Analisis

```
-- Obtener todas las sesiones de un usuario con detalles
SELECT
    rs.duration\_minutes,
    rs.distance\_km,
    rs.pace\_min\_km,
    rs.created\_at,
    (rs.distance\_km / rs.duration\_minutes * 60) as velocidad\_kmh
FROM public.running\_sessions rs
WHERE rs.user\_id = '123e4567-e89b-12d3-a456-426614174000'
ORDER BY rs.created\_at DESC;

-- Obtener estadisticas resumidas de un usuario
SELECT
    COUNT(*) as total\_sesiones,
    SUM(distance\_km) as distancia\_total,
    SUM(duration\_minutes) as tiempo\_total,
    AVG(pace\_min\_km) as ritmo\_promedio,
    MIN(pace\_min\_km) as mejor\_ritmo,
    MAX(distance\_km) as carrera\_mas\_larga
FROM public.running\_sessions
WHERE user\_id = '123e4567-e89b-12d3-a456-426614174000';

-- Obtener progreso mensual
SELECT
    DATE\_TRUNC('month', created\_at) as mes,
    COUNT(*) as sesiones\_mes,
    SUM(distance\_km) as distancia\_mes,
    AVG(pace\_min\_km) as ritmo\_promedio\_mes
FROM public.running\_sessions
WHERE user\_id = '123e4567-e89b-12d3-a456-426614174000'
GROUP BY DATE\_TRUNC('month', created\_at)
ORDER BY mes DESC;

-- Obtener las 5 mejores sesiones por ritmo
SELECT
    distance\_km,
    duration\_minutes,
    pace\_min\_km,
    created\_at
FROM public.running\_sessions
WHERE user\_id = '123e4567-e89b-12d3-a456-426614174000'
  AND pace\_min\_km IS NOT NULL
ORDER BY pace\_min\_km ASC
LIMIT 5;

-- Obtener tendencia de mejora en el ultimo mes
SELECT
    DATE\_TRUNC('week', created\_at) as semana,
    AVG(pace\_min\_km) as ritmo\_promedio\_semana
```

```
FROM public.running\_sessions
WHERE user\_id = '123e4567-e89b-12d3-a456-426614174000'
  AND created\_at >= NOW() - INTERVAL '1 month'
  AND pace\_min\_km IS NOT NULL
GROUP BY DATE\_TRUNC('week', created\_at)
ORDER BY semana ASC;
```

Listing 17: Consultas utiles para analisis de running

11.7 Flujos de Datos Completos

11.7.1 Flujo de Creacion de Sesion

1. **Usuario completa formulario** en `RunningForm`
2. **Validacion en cliente** de campos requeridos (duracion y distancia)
3. **Envio a Server Action** `createRunningSession`
4. **Verificacion de autenticacion** en servidor
5. **Calculo automatico de ritmo** si no se proporciona
6. **Insercion en base de datos** tabla `running_sessions`
7. **Revalidacion de cache** con `revalidatePath`
8. **Actualizacion de UI** con nueva sesion y estadisticas

11.7.2 Flujo de Calculo de Estadisticas

1. **Carga de sesiones** con `getRunningSessions`
2. **Calculo de metricas basicas** (total sesiones, distancia, tiempo)
3. **Calculo de ritmo promedio** de sesiones con ritmo registrado
4. **Identificacion del mejor ritmo** (valor minimo)
5. **Identificacion de la carrera mas larga** (distancia maxima)
6. **Formateo de datos** para presentacion
7. **Renderizado de estadisticas** en `RunningStats`

12 Caracteristicas Avanzadas

12.1 Calculos Automaticos

El modulo implementa varios calculos automaticos:

- **Ritmo por Kilometro**: `duracion_minutos / distancia_km`
- **Velocidad en km/h**: `distancia_km / (duracion_minutos / 60)`

- **Ritmo Promedio**: Promedio de todos los ritmos registrados
- **Mejor Ritmo**: Valor minimo de ritmo (mas rapido)
- **Distancia Total**: Suma de todas las distancias
- **Tiempo Total**: Suma de todos los tiempos

12.2 Validaciones y Restricciones

- **Duracion**: Debe ser mayor a 0 minutos
- **Distancia**: Debe ser mayor a 0 kilometros
- **Ritmo**: Opcional, se calcula automaticamente si no se proporciona
- **Usuario**: Debe estar autenticado para todas las operaciones
- **RLS**: Solo se puede acceder a sesiones propias

12.3 Formateo de Datos

- **Tiempo**: Convierte minutos a formato "Xh Ym" o "Ym"
- **Ritmo**: Convierte decimal a formato "X:YY" (minutos:segundos)
- **Distancia**: Muestra con 1 decimal (ej: "5.0 km")
- **Fechas**: Formato localizado (ej: "14/10/2025")

13 Mejores Practicas de Desarrollo

13.1 Seguridad

- **Validacion de entrada**: Siempre validar duracion y distancia
- **Autenticacion**: Verificar usuario en cada operacion
- **RLS**: Usar Row Level Security en todas las tablas
- **Sanitizacion**: Limpiar y convertir datos de entrada
- **Logs de seguridad**: Registrar operaciones sensibles

13.2 Performance

- **Indices de base de datos**: Optimizar consultas por usuario y fecha
- **Cache**: Usar revalidatePath para actualizar cache
- **Calculos eficientes**: Optimizar calculos de estadisticas
- **Lazy loading**: Cargar graficos bajo demanda
- **Paginacion**: Implementar para listas largas de sesiones

13.3 Mantenibilidad

- **TypeScript**: Usar tipado estricto en todos los componentes
- **Interfaces**: Definir interfaces claras para props
- **Separacion de responsabilidades**: Logica en Server Actions
- **Reutilizacion**: Componentes modulares y reutilizables
- **Documentacion**: Comentar codigo complejo

14 Solucion de Problemas

14.1 Problemas Comunes

14.1.1 Error: Tabla no existe

Sintomas: Error al intentar insertar o consultar datos **Causa**: Las tablas no han sido creadas en la base de datos **Solucion**: Ejecutar los scripts SQL en orden:

1. `01-create-database-schema.sql`
2. `01-create-user-schema.sql`
3. `02-create-history-schema.sql`

14.1.2 Error: Usuario no autenticado

Sintomas: Server Actions retornan error de autenticacion **Causa**: Usuario no esta logueado o sesion expirada **Solucion**: Verificar estado de autenticacion y redirigir a login

14.1.3 Error: Calculo de ritmo incorrecto

Sintomas: El ritmo calculado no coincide con el esperado **Causa**: Division por cero o valores invalidos **Solucion**: Validar que distancia sea mayor a 0 antes del calculo

14.2 Debugging

14.2.1 Logs del Cliente

```
// Habilitar logs detallados
localStorage.setItem('debug', 'true');

// Verificar estado de autenticacion
console.log('User:', user);
console.log('Session:', session);

// Verificar datos de sesiones
console.log('Running Sessions:', sessions);
```

Listing 18: Debugging en cliente

14.2.2 Logs del Servidor

```
// En Server Actions
console.log('Action called with:', { userId, data });

// En consultas de base de datos
console.log('Query result:', { data, error });

// En calculos de ritmo
console.log('Pace calculation:', { duration, distance, calculatedPace })
    ;
```

Listing 19: Debugging en servidor

15 Conclusion

El modulo de running de FitTrack es un sistema completo y robusto que permite a los usuarios gestionar sus sesiones de carrera de manera eficiente. Con su arquitectura bien definida, base de datos optimizada y componentes React modernos, proporciona una experiencia de usuario excepcional.

Caracteristicas destacadas:

- Arquitectura escalable y mantenible
- Seguridad robusta con RLS
- Interfaz de usuario intuitiva
- Calculos automaticos precisos
- Estadisticas detalladas y utiles
- Codigo bien documentado y tipado

Para contribuir al desarrollo del modulo:

1. Seguir las convenciones establecidas
2. Implementar tests apropiados
3. Documentar cambios significativos
4. Mantener la compatibilidad con la base de datos
5. Respetar las politicas de seguridad

16 Módulo de Comidas

Ubicación: `app/meals/` y `components/meals/`

17 Módulo de Asistente Nutricional (IA)

El módulo de Asistente Nutricional es uno de los componentes más avanzados y tecnológicamente sofisticados de FitTrack. Integra inteligencia artificial de Google Gemini para proporcionar asistencia nutricional personalizada, análisis de imágenes de comida y recomendaciones fitness basadas en el progreso real del usuario.

17.1 Características del Módulo de Asistente Nutricional

- **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

17.2 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

18 Estructura de Archivos

18.1 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 20: Estructura de archivos del Asistente Fitness

19 APIs y Endpoints

19.1 API de Chat - /api/chat

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

19.1.1 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 21: Estructura basica del endpoint /api/chat

19.1.2 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) {
```

```
7       exerciseData = `\n\nTus ejercicios registrados: ${exercises.join
    (", ")}`
8
9       const lastExercise = exercises[0]
10      const history = await getExerciseHistory(lastExercise, 30)
11      if (history.length > 0) {
12        const latest = history[0]
13        exerciseData += `\n\nUltimo registro de ${lastExercise}: ${
    latest.weight_kg}kg x ${latest.repetitions} reps`
14      }
15    }
16  } catch (error) {
17    console.log("No se pudo obtener historial de ejercicios")
18  }
19
20  // 2. Datos de running
21  let runningData = ""
22  try {
23    const runningSessions = await getRunningHistory(30)
24    if (runningSessions.length > 0) {
25      const totalDistance = runningSessions.reduce((sum, session) => sum
    + session.distance, 0)
26      const avgPace = runningSessions.reduce((sum, session) => sum +
    session.pace, 0) / runningSessions.length
27      const lastSession = runningSessions[0]
28
29      runningData = `\n\nDATOS DE RUNNING (ultimos 30 dias):
30  - Total de sesiones: ${runningSessions.length}
31  - Distancia total: ${totalDistance.toFixed(2)}km
32  - Pace promedio: ${avgPace.toFixed(2)} min/km
33  - Ultima sesion: ${lastSession.distance}km en ${lastSession.duration}
    minutos (${lastSession.pace.toFixed(2)} min/km)`
34    }
35  } catch (error) {
36    console.log("No se pudo obtener historial de running")
37  }
38
39  // 3. Datos de gimnasio
40  let gymData = ""
41  try {
42    const workouts = await getWorkouts()
43    if (workouts.length > 0) {
44      const recentWorkouts = workouts.slice(0, 5)
45      gymData = `\n\nULTIMOS ENTRENAMIENTOS EN GIMNASIO:`
46      recentWorkouts.forEach((workout: any) => {
47        const date = new Date(workout.created_at).toLocaleDateString()
48        gymData += `\n- ${workout.exercise_name}: ${workout.weight_kg ||
    0}kg x ${workout.repetitions || 0} reps x ${workout.sets || 0} sets
    (${date})`
49      })
50    }
51  } catch (error) {
52    console.log("No se pudo obtener workouts del gimnasio")
53  }
54
55  // 4. Calculos nutricionales personalizados
56  let bmr = 0, tdee = 0, proteinMin = 0, proteinMax = 0
57
```

```
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 22: Funcion handleChatMessage - Chat personalizado

19.1.3 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 })
```

```
47   }
48 }
```

Listing 23: Funcion handleImageAnalysis - Analisis con IA

19.2 API de Analisis - /api/analyze

Endpoint especializado para el analisis detallado de imagenes de comida.

19.3 APIs del Módulo de Asistente Nutricional

19.3.1 Características Principales

- **Análisis de Comidas por Imagen**: Los usuarios pueden subir una foto de su comida y recibir un análisis nutricional detallado, incluyendo calorías, macronutrientes e ingredientes.
- **Chat Conversacional**: Un chatbot inteligente que responde a preguntas sobre nutrición, sugiere recetas y ofrece consejos personalizados basados en el perfil del usuario.
- **Personalización Avanzada**: Las respuestas y análisis de la IA se adaptan al perfil del usuario (peso, altura, objetivos, etc.) para ofrecer recomendaciones más precisas.
- **Integración de Múltiples Modelos de IA**: Utiliza ‘Gemini 1.5 Flash‘ para el chat y ‘Gemini 1.5 Pro‘ para el análisis de imágenes, seleccionando el modelo más adecuado para cada tarea.

19.3.2 Arquitectura de la API

La lógica de la IA se expone a través de dos API Routes principales que se comunican con los modelos de Google Gemini.

- `/api/chat`: Gestiona las conversaciones del chatbot. Recibe el mensaje del usuario y el perfil de salud, construye un *prompt* contextualizado y devuelve la respuesta generada por el modelo de texto de Gemini.
- `/api/analyze-image`: Maneja el análisis de imágenes de comidas. Recibe una imagen en formato base64, la envía al modelo de visión de Gemini con un *prompt* estructurado que solicita un análisis en formato JSON, y devuelve los datos nutricionales extraídos.

```
1 // POST /api/chat
2 export async function POST(request: Request) {
3   // 1. Extraer el mensaje y el perfil del usuario del body.
4   const { message, userProfile } = await request.json();
5
6   // 2. Construir un ‘systemPrompt‘ que instruye a la IA sobre su rol
      como nutricionista.
7   //    Se personaliza el prompt con los datos del ‘userProfile‘ (IMC,
      BMR, etc.).
8   const systemPrompt = buildSystemPrompt(userProfile);
9
```

```
  // 3. Llamar al modelo de texto de Gemini con el prompt.
  const model = google("gemini-1.5-flash");
  const { text } = await generateText({ model, prompt: [systemPrompt,
   message] });

  // 4. Devolver la respuesta de la IA.
  return NextResponse.json({ response: text });
}
```

Listing 24: Lógica simplificada del endpoint ‘/api/chat‘

19.3.3 Componentes Principales de React

La interfaz de usuario se divide en componentes especializados para cada funcionalidad.

- `app/meals/page.tsx`: Es la página principal del módulo. Utiliza un sistema de pestañas (*Tabs*) para permitir al usuario cambiar entre el "Chat con IA" y el "Analizador de Comida".
- `components/meals/chat-interface.tsx`: Implementa la interfaz del chatbot. Gestiona el historial de la conversación, el estado de carga y la obtención del perfil del usuario para enriquecer las consultas a la API. Utiliza `useActionState` para manejar el envío de mensajes.
- `components/meals/image-analyzer.tsx`: Permite al usuario subir o capturar una imagen de su comida. Gestiona el estado de la imagen (vista previa, carga), llama al endpoint `/api/analyze-image` y muestra los resultados del análisis nutricional en un formato claro y estructurado.

```
"use client"

import type React from "react"
import { useState, useRef, useEffect } from "react"
import { Button } from "@/components/ui/button"
import { Card, CardContent, CardHeader, CardTitle } from "@/components/
   ui/card"
import { Badge } from "@/components/ui/badge"
import { Camera, Upload, X, Sparkles, User, Loader2 } from "lucide-react
   "
import { getUserProfile } from "@/lib/user-actions"
import { calculateBMI } from "@/lib/health-actions"

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
   )
```

```
24  const [analysis, setAnalysis] = useState<string | null>(null)
25  const [isAnalyzing, setIsAnalyzing] = useState(false)
26  const [isClient, setIsClient] = useState(false)
27  const [userProfile, setUserProfile] = useState<UserProfile | null>(
     null)
28  const fileInputRef = useRef<HTMLInputElement>(null)
29  const cameraInputRef = useRef<HTMLInputElement>(null)
30
31  useEffect(() => {
32    setIsClient(true)
33    loadUserProfile()
34  }, [])
35
36  const loadUserProfile = async () => {
37    try {
38      const profile = await getUserProfile()
39      if (profile) {
40        let age: number | undefined
41        let bmi: number | undefined
42        let bmiCategory: string | undefined
43
44        if (profile.dateOfBirth) {
45          const birthDate = new Date(profile.dateOfBirth)
46          age = new Date().getFullYear() - birthDate.getFullYear()
47        }
48
49        if (profile.weight && profile.height) {
50          const bmiResult = await calculateBMI(profile.weight, profile.
     height, age, profile.sex || undefined)
51          bmi = bmiResult.bmi
52          bmiCategory = bmiResult.category
53        }
54
55        setUserProfile({
56          weight: profile.weight,
57          height: profile.height,
58          dateOfBirth: profile.dateOfBirth,
59          sex: profile.sex,
60          bmi,
61          bmiCategory,
62          age,
63        })
64      }
65    } catch (error) {
66      console.error("Error loading user profile:", error)
67    }
68  }
69
70  const handleFileSelect = (event: React.ChangeEvent<HTMLInputElement>)
     => {
71    const file = event.target.files?.[0]
72    if (file) {
73      const reader = new FileReader()
74      reader.onloadend = () => {
75        setSelectedImage(reader.result as string)
76        setAnalysis(null)
77      }
78      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 = ""
  }

  // Resto del componente...
}
```

Listing 25: 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

20 Integracion con IA

20.1 Google Gemini Integration

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

20.1.1 Configuracion de API

```
// Variables de entorno requeridas
GOOGLE_GENERATIVE_AI_API_KEY=tu_clave_de_gemini_aqui
GEMINI_API_KEY=tu_clave_de_gemini_aqui

// Importaciones necesarias
import { generateText, generateObject } from "ai"
import { google } from "@ai-sdk/google"
import { GoogleGenerativeAI } from "@google/generative-ai"
```

Listing 26: Configuracion de Google Gemini

20.1.2 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

20.2 Schemas de Validacion

20.2.1 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 27: Schema Zod para validacion de analisis

21 Calculos Nutricionales

21.1 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 28: Calculo de BMR personalizado

21.2 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 29: Calculo de proteinas recomendadas

22 Integracion con Datos del Usuario

22.1 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 30: Integracion con datos de actividad

22.2 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

23 Flujos de Datos

23.1 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

23.2 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)

24 Caracteristicas Avanzadas

24.1 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 31: Temas rapidos del chat

24.2 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

24.3 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 32: Validacion de tipos de archivo

25 Mejores Practicas de Desarrollo

25.1 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

25.2 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

25.3 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

26 Solucion de Problemas

26.1 Problemas Comunes

26.1.1 Error: API key no configurada

Sintomas: Error "API key no configurada" en respuestas **Causa**: Variables de entorno no configuradas **Solucion**:

1. Crear archivo `.env.local`
2. Agregar `GOOGLE_GENERATIVE_AI_API_KEY=tu_clave`
3. Reiniciar servidor de desarrollo

26.1.2 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

26.1.3 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

26.2 Debugging

26.2.1 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 33: Debugging en cliente

26.2.2 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 34: Debugging en servidor

27 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

28 Módulo de Salud

Ubicación: `app/health/` y `components/health/`

28.1 Características del Módulo de Salud

- Calculadora de IMC
- Registro de signos vitales
- Evaluación automática de salud
- Historial de métricas

29 Módulo de Mensajería

El módulo de mensajería de FitTrack es un sistema integral que combina comunicación en tiempo real entre usuarios y profesionales con un sistema robusto de gestión de roles y configuraciones avanzadas de accesibilidad. Este módulo está diseñado para facilitar la interacción entre usuarios regulares y profesionales de la salud y fitness, mientras garantiza la inclusividad y accesibilidad para todos los usuarios.

29.1 Características del Módulo de Mensajería

- **Sistema de Mensajería en Tiempo Real**: Comunicación instantánea entre usuarios y profesionales
- **Gestión de Roles Avanzada**: Sistema de tres niveles (Usuario, Profesional, Administrador)
- **Configuraciones de Accesibilidad**: Soporte completo para usuarios con diferentes necesidades
- **Panel de Administración**: Gestión centralizada de usuarios y roles
- **Seguridad Robusta**: Row Level Security (RLS) y validaciones estrictas
- **Interfaz Intuitiva**: Diseño responsivo y accesible
- **Notificaciones en Tiempo Real**: Actualizaciones automáticas de mensajes

29.2 Arquitectura General

El módulo sigue una arquitectura de microservicios con separación clara de responsabilidades:

1. **Capa de Presentación**: Componentes React con TypeScript
2. **Capa de Lógica**: Server Actions de Next.js
3. **Capa de Datos**: Supabase PostgreSQL con RLS
4. **Capa de Servicios**: Gestión de roles y accesibilidad
5. **Capa de Seguridad**: Autenticación y autorización

30 Estructura de Base de Datos

30.1 Diagrama de Relaciones

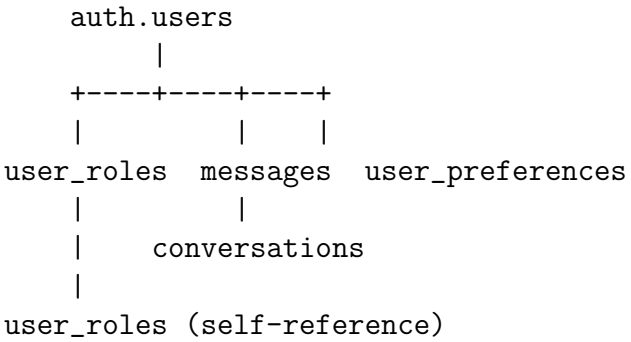

Figura 2: Diagrama de relaciones de las tablas del módulo de mensajería

30.2 Tabla user_roles

Esta tabla gestiona los roles y permisos de todos los usuarios del sistema.

```
CREATE TABLE IF NOT EXISTS public.user\_roles (
  id UUID DEFAULT gen\_random\_uuid() PRIMARY KEY,
  user\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL
   UNIQUE,
  role TEXT NOT NULL CHECK (role IN ('user', 'professional', 'admin')),
  is\_active BOOLEAN DEFAULT true,
  is\_professional BOOLEAN DEFAULT false,
  approved\_by UUID REFERENCES auth.users(id),
  approved\_at TIMESTAMP WITH TIME ZONE,
  created\_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, NOW
   ()) NOT NULL,
  updated\_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, NOW
   ()) NOT NULL
);
```

Listing 35: Estructura completa de user_roles

Descripción de campos:

- `id`: Identificador único UUID generado automáticamente
- `user_id`: Referencia al usuario en auth.users
- `role`: Rol del usuario ('user', 'professional', 'admin')
- `is_active`: Estado activo/inactivo del usuario
- `is_professional`: Indica si el usuario es profesional aprobado
- `approved_by`: ID del administrador que aprobó el rol
- `approved_at`: Timestamp de aprobación
- `created_at`: Timestamp de creación
- `updated_at`: Timestamp de última modificación

30.3 Tabla messages

Almacena todos los mensajes del sistema de mensajería.

```
CREATE TABLE IF NOT EXISTS public.messages (
  id UUID DEFAULT gen\_random\_uuid() PRIMARY KEY,
  sender\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  receiver\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL
   ,
  content TEXT NOT NULL,
  read BOOLEAN DEFAULT false,
  created\_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, NOW
   ()) NOT NULL
);
```

Listing 36: Estructura completa de messages

Descripción de campos:

- `id`: Identificador único del mensaje
- `sender_id`: ID del usuario que envía el mensaje
- `receiver_id`: ID del usuario que recibe el mensaje
- `content`: Contenido del mensaje
- `read`: Estado de lectura del mensaje
- `created_at`: Timestamp de envío

30.4 Tabla conversations

Agrupa los mensajes en conversaciones para facilitar la gestión.

```
CREATE TABLE IF NOT EXISTS public.conversations (
  id UUID DEFAULT gen\_random\_uuid() PRIMARY KEY,
  user1\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  user2\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  last\_message\_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::
   text, NOW()) NOT NULL,
  created\_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, NOW
   ()) NOT NULL,
  UNIQUE(user1\_id, user2\_id)
);
```

Listing 37: Estructura completa de conversations

Descripción de campos:

- `id`: Identificador único de la conversación
- `user1_id`: ID del primer usuario (ordenado alfabéticamente)
- `user2_id`: ID del segundo usuario (ordenado alfabéticamente)
- `last_message_at`: Timestamp del último mensaje
- `created_at`: Timestamp de creación de la conversación

30.5 Tabla user_preferences

Almacena las configuraciones de accesibilidad de cada usuario.

```
CREATE TABLE IF NOT EXISTS public.user\_preferences (
  id UUID DEFAULT gen\_random\_uuid() PRIMARY KEY,
  user\_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL
   UNIQUE,
  color\_blind\_mode TEXT CHECK (color\_blind\_mode IN ('none', '
   protanopia', 'deuteranopia', 'tritanopia')),
  high\_contrast BOOLEAN DEFAULT false,
  large\_text BOOLEAN DEFAULT false,
  reduce\_motion BOOLEAN DEFAULT false,
  screen\_reader\_optimized BOOLEAN DEFAULT false,
  created\_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, NOW
   ()) NOT NULL,
```

```
  updated\_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text, NOW
    ()) NOT NULL
);
```

Listing 38: Estructura completa de user_preferences

Descripción de campos:

- `id`: Identificador único de las preferencias
- `user_id`: Referencia al usuario propietario
- `color_blind_mode`: Modo de daltonismo configurado
- `high_contrast`: Activar alto contraste
- `large_text`: Activar texto grande
- `reduce_motion`: Reducir animaciones
- `screen_reader_optimized`: Optimización para lectores de pantalla
- `created_at`: Timestamp de creación
- `updated_at`: Timestamp de última modificación

30.6 Políticas de Seguridad (RLS)

Todas las tablas implementan Row Level Security para garantizar la privacidad y seguridad de los datos.

```
-- Habilitar RLS
ALTER TABLE public.user\_roles ENABLE ROW LEVEL SECURITY;

-- Usuarios pueden ver su propio rol
CREATE POLICY "Users can view own role" ON public.user\_roles
  FOR SELECT USING (auth.uid() = user\_id);

-- Administradores pueden ver todos los roles
CREATE POLICY "Admins can view all roles" ON public.user\_roles
  FOR SELECT USING (
    EXISTS (
      SELECT 1 FROM public.user\_roles
      WHERE user\_id = auth.uid() AND role = 'admin'
    )
  );

-- Administradores pueden actualizar roles
CREATE POLICY "Admins can update roles" ON public.user\_roles
  FOR UPDATE USING (
    EXISTS (
      SELECT 1 FROM public.user\_roles
      WHERE user\_id = auth.uid() AND role = 'admin'
    )
  );
```

Listing 39: Políticas RLS para user_roles

```
-- Habilitar RLS
ALTER TABLE public.messages ENABLE ROW LEVEL SECURITY;

-- Usuarios pueden ver sus propios mensajes
CREATE POLICY "Users can view own messages" ON public.messages
  FOR SELECT USING (auth.uid() = sender\_id OR auth.uid() = receiver\_id
    );

-- Usuarios pueden enviar mensajes
CREATE POLICY "Users can send messages" ON public.messages
  FOR INSERT WITH CHECK (auth.uid() = sender\_id);

-- Usuarios pueden marcar mensajes como leidos
CREATE POLICY "Users can update own messages" ON public.messages
  FOR UPDATE USING (auth.uid() = receiver\_id);
```

Listing 40: Políticas RLS para messages

31 Módulo de Administrador

31.1 Base de Datos

El sistema utiliza las siguientes tablas principales:

31.1.1 Tabla user_roles

```
  CREATE TABLE user_roles (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,
  role TEXT NOT NULL CHECK (role IN ('user', 'admin')),
  is_active BOOLEAN DEFAULT true,
  is_professional BOOLEAN DEFAULT false,
  approved_by UUID REFERENCES auth.users(id),
  approved_at TIMESTAMPTZ,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
  );
```

Listing 41: Estructura de la tabla user_roles

31.1.2 Tabla gym_exercises

```
  CREATE TABLE gym_exercises (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name TEXT NOT NULL,
  category TEXT NOT NULL CHECK (category IN ('Pecho', 'Biceps', 'Triceps
    ', 'Hombros', 'Pierna', 'Espalda', 'Otros')),
  description TEXT,
  image_url TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
  );
```

Listing 42: Estructura de la tabla gym_exercises

32 Sistema de Autenticación y Autorización

32.1 Verificación de Privilegios de Administrador

La función `isAdmin()` es el núcleo del sistema de autorización:

```
export async function isAdmin() {
  try {
    const supabase = await createClient()
    const { data: { user }, error: userError } = await supabase.auth.
 getUser()

    if (userError || !user) {
      return false
    }

    const { data, error } = await supabase
    .from("user_roles")
    .select("role")
    .eq("user_id", user.id)
    .maybeSingle()

    if (error || !data) {
      return false
    }

    return data.role === "admin"
  } catch (error) {
    console.error("[v0] Exception in isAdmin:", error)
    return false
  }
}
```

Listing 43: Función isAdmin en admin-actions.ts

32.2 Protección de Rutas

Cada página de administración implementa protección de acceso:

```
export default async function AdminPage() {
  const admin = await isAdmin()

  if (!admin) {
    redirect("/")
  }
  // ... resto del componente
}
```

Listing 44: Protección de rutas en page.tsx

33 Funcionalidades del Dashboard Principal

33.1 Componente AdminDashboard

El componente principal del dashboard ofrece:

1. **Estadísticas del Sistema**:
   - Total de usuarios registrados
   - Número de profesionales registrados
   - Cantidad de administradores
2. **Gestión de Usuarios**:
   - Lista completa de usuarios con información detallada
   - Cambio de roles (Usuario/Admin)
   - Activación/desactivación de usuarios
   - Marcado de usuarios como profesionales
   - Búsqueda y filtrado de usuarios
3. **Interfaz de Usuario**:
   - Diseño responsivo con Tailwind CSS
   - Componentes de UI modernos (shadcn/ui)
   - Notificaciones toast para feedback
   - Estados de carga durante operaciones

33.2 Estados y Gestión de Datos

El dashboard utiliza React hooks para el manejo de estado:

```
const [users, setUsers] = useState(initialUsers)
const [loading, setLoading] = useState<string | null>(null)
const [searchQuery, setSearchQuery] = useState("")
const { toast } = useToast()
```

Listing 45: Estados del componente AdminDashboard

33.3 Funciones de Gestión de Usuarios

33.3.1 Cambio de Roles

```
const handleRoleChange = async (userId: string, role: string) => {
  setLoading(userId)
  const user = users.find((u) => u.id === userId)
  if (!user) return

  const result = await updateUserRole(userId, role, user.is_active,
 user.is_professional)

  if (result.error) {
    toast({
      title: "Error",
      description: result.error,
      variant: "destructive",
    })
  } else {
```

```
      toast({
        title: "Exito",
        description: "Rol actualizado correctamente",
      })
      setUsers(users.map((u) => (u.id === userId ? { ...u, role } : u)))
    }
    setLoading(null)
  }
```

Listing 46: Función handleRoleChange

34 Gestión de Ejercicios

34.1 Componente ExerciseManagement

El sistema de gestión de ejercicios permite a los administradores:

1. **Crear Nuevos Ejercicios**:
   - Nombre del ejercicio
   - Categoría (Pecho, Bíceps, Tríceps, Hombros, Pierna, Espalda, Otros)
   - Descripción opcional
   - Imagen del ejercicio (URL externa o subida de archivo)
2. **Editar Ejercicios Existentes**:
   - Modificación de todos los campos
   - Actualización de imágenes
   - Preservación del historial
3. **Eliminar Ejercicios**:
   - Eliminación con confirmación
   - Limpieza de referencias
4. **Búsqueda y Filtrado**:
   - Búsqueda por nombre
   - Filtrado por categoría
   - Estadísticas por categoría

34.2 Sistema de Imágenes

El sistema soporta dos métodos para manejar imágenes:

1. **URL Externa**: Los administradores pueden proporcionar URLs de imágenes externas
2. **Subida de Archivos**: Sistema de subida a Supabase Storage con:

- Validación de tipo de archivo (solo imágenes)
- Límite de tamaño (5MB máximo)
- Generación automática de nombres únicos
- Vista previa en tiempo real

34.3 Función de Subida de Imágenes

```
export async function uploadExerciseImage(formData: FormData) {
  const file = formData.get("file") as File

  if (!file) {
    return { error: "No se proporciono ningun archivo" }
  }

  if (!file.type.startsWith("image/")) {
    return { error: "El archivo debe ser una imagen" }
  }

  if (file.size > 5 * 1024 * 1024) {
    return { error: "El archivo no debe superar los 5MB" }
  }

  try {
    const supabase = await createClient()
    const { data: { user }, error: userError } = await supabase.auth.
getUser()

    if (userError || !user) {
      return { error: "Usuario no autenticado" }
    }

    const fileExt = file.name.split(".").pop()
    const fileName = `${Date.now()}-${Math.random().toString(36).
substring(7)}.${fileExt}`
    const filePath = `exercise-images/${fileName}`

    const { data, error: uploadError } = await supabase.storage
    .from("exercise-images")
    .upload(filePath, file, {
      cacheControl: "3600",
      upsert: false,
    })

    if (uploadError) {
      return { error: "Error al subir la imagen" }
    }

    const { data: { publicUrl } } = supabase.storage
    .from("exercise-images")
    .getPublicUrl(filePath)

    return { success: true, imageUrl: publicUrl }
  } catch (error) {
    return { error: "Error de conexion con la base de datos" }
  }
```

```
}
```

Listing 47: Función uploadExerciseImage

35 Funciones de Base de Datos

35.1 Función get_all_users_with_roles

Esta función permite a los administradores obtener información completa de todos los usuarios:

```
CREATE OR REPLACE FUNCTION get_all_users_with_roles()
RETURNS TABLE (
id uuid,
email text,
full_name text,
role text,
is_active boolean,
is_professional boolean,
created_at timestamptz
)
SECURITY DEFINER
SET search_path = public
LANGUAGE plpgsql
AS $$
BEGIN
-- Check if the current user is an admin
IF NOT EXISTS (
SELECT 1 FROM user_roles ur_check
WHERE ur_check.user_id = auth.uid()
AND ur_check.role = 'admin'
) THEN
RAISE EXCEPTION 'No autorizado';
END IF;

-- Return all users with their roles
RETURN QUERY
SELECT
au.id,
au.email::text,
COALESCE(
au.raw_user_meta_data->>'full_name',
CONCAT(
COALESCE(au.raw_user_meta_data->>'first_name', ''),
' ',
COALESCE(au.raw_user_meta_data->>'last_name', '')
),
'Sin nombre'
)::text,
COALESCE(ur.role, 'user')::text,
COALESCE(ur.is_active, true),
COALESCE(ur.is_professional, false),
au.created_at
FROM auth.users au
LEFT JOIN user_roles ur ON ur.user_id = au.id
ORDER BY au.created_at DESC;
END;
```

```
$$;
```

Listing 48: Función get_all_users_with_roles

35.2 Función is_admin

Función auxiliar para verificar privilegios administrativos:

```
CREATE OR REPLACE FUNCTION public.is_admin(check_user_id UUID)
RETURNS BOOLEAN AS $$
DECLARE
user_role TEXT;
BEGIN
SELECT role INTO user_role
FROM public.user_roles
WHERE user_id = check_user_id;

RETURN user_role = 'admin';
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
```

Listing 49: Función is_admin

36 Seguridad y Row Level Security (RLS)

36.1 Políticas de Seguridad

El sistema implementa múltiples capas de seguridad:

1. **RLS en user_roles**:

```
    -- Admins can view all roles
    CREATE POLICY "Admins can view all roles" ON public.user_roles
    FOR SELECT USING (public.is_admin(auth.uid()));

    -- Admins can update roles
    CREATE POLICY "Admins can update roles" ON public.user_roles
    FOR UPDATE USING (public.is_admin(auth.uid()));

    -- Admins can insert roles
    CREATE POLICY "Admins can insert roles" ON public.user_roles
    FOR INSERT WITH CHECK (public.is_admin(auth.uid()));

```

Listing 50: Políticas RLS para user_roles

2. **RLS en gym_exercises**:

```
    -- Anyone can view gym exercises
    CREATE POLICY "Anyone can view gym exercises"
    ON gym_exercises
    FOR SELECT
    TO authenticated
    USING (true);

    -- Only admins can manage gym exercises
```

```
    CREATE POLICY "Only admins can manage gym exercises"
    ON gym_exercises
    FOR ALL
    TO authenticated
    USING (
    EXISTS (
    SELECT 1 FROM user_roles
    WHERE user_roles.user_id = auth.uid()
    AND user_roles.role = 'admin'
    )
    );

```

Listing 51: Políticas RLS para gym_exercises

36.2 Funciones SECURITY DEFINER

Las funciones marcadas como `SECURITY DEFINER` ejecutan con los privilegios del propietario de la función, permitiendo operaciones que de otro modo estarían restringidas por RLS.

37 Scripts de Configuración

37.1 Creación del Usuario Administrador

Para establecer un usuario administrador, se debe ejecutar el siguiente script:

```
-- Insert admin role for the admin user
-- First, create user through Supabase Auth with email: juan@ejemplo.
 com and password: 123456
-- Then run this script to assign the admin role

INSERT INTO public.user_roles (user_id, role, is_active, approved_at)
SELECT id, 'admin', true, NOW()
FROM auth.users
WHERE email = 'juan@ejemplo.com'
ON CONFLICT (user_id) DO UPDATE
SET role = 'admin', is_active = true, approved_at = NOW();

-- Verify the admin was created
SELECT u.id, u.email, ur.role, ur.is_active
FROM auth.users u
LEFT JOIN public.user_roles ur ON u.id = ur.user_id
WHERE u.email = 'juan@ejemplo.com';
```

Listing 52: Script para crear usuario administrador

37.2 Inserción de Ejercicios por Defecto

El sistema incluye ejercicios predefinidos para todas las categorías:

```
INSERT INTO gym_exercises (name, category, description) VALUES
-- Pecho
('Press de Banca', 'Pecho', 'Ejercicio basico para pecho con barra'),
('Press Inclinado', 'Pecho', 'Press de banca en banco inclinado'),
```

```
('Aperturas con Mancuernas', 'Pecho', 'Aperturas para pecho'),
('Fondos en Paralelas', 'Pecho', 'Fondos para pecho y triceps'),

-- Biceps
('Curl con Barra', 'Biceps', 'Curl de biceps con barra recta'),
('Curl con Mancuernas', 'Biceps', 'Curl alternado con mancuernas'),
('Curl Martillo', 'Biceps', 'Curl con agarre neutro'),
('Curl en Banco Scott', 'Biceps', 'Curl concentrado en banco'),

-- Triceps
('Press Frances', 'Triceps', 'Extension de triceps acostado'),
('Fondos para Triceps', 'Triceps', 'Fondos en banco'),
('Extension en Polea', 'Triceps', 'Extension de triceps en polea alta'
 ),
('Patada de Triceps', 'Triceps', 'Extension con mancuerna'),

-- Hombros
('Press Militar', 'Hombros', 'Press de hombros con barra'),
('Elevaciones Laterales', 'Hombros', 'Elevaciones laterales con
 mancuernas'),
('Elevaciones Frontales', 'Hombros', 'Elevaciones frontales'),
('Pajaros', 'Hombros', 'Elevaciones posteriores'),

-- Pierna
('Sentadilla', 'Pierna', 'Sentadilla con barra'),
('Prensa de Pierna', 'Pierna', 'Press de piernas en maquina'),
('Peso Muerto', 'Pierna', 'Peso muerto convencional'),
('Zancadas', 'Pierna', 'Zancadas con mancuernas'),
('Extension de Cuadriceps', 'Pierna', 'Extension en maquina'),
('Curl Femoral', 'Pierna', 'Curl de piernas acostado'),
('Elevacion de Gemelos', 'Pierna', 'Elevacion de pantorrillas'),

-- Espalda
('Dominadas', 'Espalda', 'Dominadas con peso corporal'),
('Remo con Barra', 'Espalda', 'Remo inclinado con barra'),
('Remo con Mancuerna', 'Espalda', 'Remo a una mano'),
('Jalon al Pecho', 'Espalda', 'Jalon en polea alta'),
('Peso Muerto Rumano', 'Espalda', 'Peso muerto para espalda baja'),

-- Otros
('Plancha', 'Otros', 'Plancha abdominal'),
('Abdominales', 'Otros', 'Crunch abdominal'),
('Cardio', 'Otros', 'Ejercicio cardiovascular');
```

Listing 53: Ejercicios por defecto

38 Integración con Otros Módulos

38.1 Sistema de Mensajería

El módulo de administrador se integra con el sistema de mensajería permitiendo:

- Gestionar qué usuarios son profesionales
- Activar/desactivar profesionales para el sistema de chat
- Controlar la visibilidad de usuarios en el sistema de mensajería

38.2 Sistema de Ejercicios del Gimnasio

Los ejercicios gestionados por el administrador están disponibles para:

- Selección en rutinas de gimnasio
- Creación de historial de ejercicios
- Seguimiento de progreso de usuarios

39 Consideraciones de Rendimiento

39.1 Índices de Base de Datos

El sistema incluye índices optimizados:

```
-- indice para consultas de categoria de ejercicios
CREATE INDEX idx_gym_exercises_category ON gym_exercises(category);

-- indice para consultas de usuarios profesionales
CREATE INDEX idx_user_roles_is_professional
ON user_roles(is_professional)
WHERE is_professional = true;
```

Listing 54: Índices para optimización

39.2 Optimizaciones de Frontend

- **Lazy Loading**: Componentes se cargan bajo demanda
- **Debouncing**: Búsquedas con retraso para reducir consultas
- **Estados Optimizados**: Actualizaciones locales del estado
- **Revalidación**: Uso de `revalidatePath` para cache

40 Testing y Debugging

40.1 Logs de Debugging

El sistema incluye logging extensivo para debugging:

```
console.log("[v0] Form data before submission:", formData)
console.log("[v0] Image URL value:", formData.image_url)
console.log("[v0] Server action result:", result)
```

Listing 55: Ejemplo de logging

40.2 Manejo de Errores

- **Try-Catch**: Captura de excepciones en todas las operaciones
- **Validación de Datos**: Verificación de entrada en cliente y servidor
- **Feedback Visual**: Notificaciones toast para todos los estados
- **Fallbacks**: Comportamiento degradado en caso de errores

41 Configuración de Desarrollo

41.1 Variables de Entorno

El módulo requiere las siguientes variables de entorno:

```
NEXT_PUBLIC_SUPABASE_URL=your_supabase_url
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key
SUPABASE_SERVICE_ROLE_KEY=your_service_role_key
```

Listing 56: Variables de entorno necesarias

41.2 Dependencias

Las principales dependencias del módulo incluyen:

```
{
  "@supabase/supabase-js": "^2.38.0",
  "@radix-ui/react-select": "^2.0.0",
  "@radix-ui/react-switch": "^1.0.3",
  "@radix-ui/react-tabs": "^1.0.4",
  "lucide-react": "^0.292.0",
  "next": "14.0.0",
  "react": "^18.0.0",
  "tailwindcss": "^3.3.0"
}
```

Listing 57: Dependencias principales

42 Despliegue y Producción

42.1 Consideraciones de Seguridad

1. **RLS Habilitado**: Todas las tablas tienen RLS activado
2. **Funciones SECURITY DEFINER**: Solo para operaciones específicas
3. **Validación de Entrada**: Sanitización en cliente y servidor
4. **Autenticación Requerida**: Todas las rutas protegidas

42.2 Monitoreo

- **Logs de Supabase**: Monitoreo de consultas y errores
- **Analytics**: Seguimiento de uso del dashboard
- **Alertas**: Notificaciones de errores críticos

43 Conclusión

El módulo de administrador de FitTrack representa una implementación robusta y segura de un sistema de gestión administrativa. Con su arquitectura bien estructurada, sistema de seguridad multicapa y interfaz de usuario moderna, proporciona las herramientas necesarias para gestionar eficientemente una aplicación de fitness.

Las características clave incluyen:

- **Seguridad**: Implementación completa de RLS y funciones de seguridad
- **Escalabilidad**: Diseño modular que permite fácil extensión
- **Usabilidad**: Interfaz intuitiva con feedback visual inmediato
- **Mantenibilidad**: Código bien documentado y estructurado
- **Rendimiento**: Optimizaciones en base de datos y frontend

Este módulo sirve como base sólida para futuras expansiones y mejoras del sistema FitTrack.

44 Server Actions - Lógica de Negocio

Las Server Actions manejan toda la lógica de negocio del módulo de mensajería. Están implementadas en TypeScript con validación robusta y manejo de errores.

44.1 Archivo messaging-actions.ts

44.1.1 sendMessage - Enviar Mensaje

```
1  "use server"
2
3  import { createClient } from "@/lib/supabase/server"
4
5  export async function sendMessage(receiverId: string, content: string) {
6    const supabase = await createClient()
7    const {
8      data: { user },
9    } = await supabase.auth.getUser()
10
11   if (!user) {
12     return { error: "No autenticado" }
13   }
14
```

```
  // Crear o actualizar conversacion
  const { error: convError } = await supabase.from("conversations").upsert(
    {
      user1\_id: user.id < receiverId ? user.id : receiverId,
      user2\_id: user.id < receiverId ? receiverId : user.id,
      last\_message\_at: new Date().toISOString(),
    },
    {
      onConflict: "user1\_id,user2\_id",
    },
  )

  if (convError) {
    console.error("Error creating conversation:", convError)
  }

  // Enviar mensaje
  const { error } = await supabase.from("messages").insert({
    sender\_id: user.id,
    receiver\_id: receiverId,
    content,
  })

  if (error) {
    return { error: error.message }
  }

  return { success: true }
}
```

Listing 58: Función sendMessage completa

Características importantes:

- **Validación de autenticación**: Verifica que el usuario esté autenticado
- **Creación automática de conversación**: Crea la conversación si no existe
- **Ordenamiento consistente**: user1_id siempre es menor que user2_id
- **Manejo de errores**: Captura y reporta errores de base de datos
- **Actualización de timestamp**: Actualiza last_message_at automáticamente

44.1.2 getMessages - Obtener Mensajes

```
export async function getMessages(otherUserId: string) {
  const supabase = await createClient()
  const {
    data: { user },
  } = await supabase.auth.getUser()

  if (!user) {
    return { error: "No autenticado" }
  }

  const { data, error } = await supabase
```

```
12      .from("messages")
13      .select("*")
14      .or(
15        `and(sender\_id.eq.${user.id},receiver\_id.eq.${otherUserId}),and(
      sender\_id.eq.${otherUserId},receiver\_id.eq.${user.id})`,
16      )
17      .order("created\_at", { ascending: true })
18
19    if (error) {
20      return { error: error.message }
21    }
22
23    return { messages: data }
24  }
```

Listing 59: Función getMessages completa

44.1.3 getAvailableContacts - Obtener Contactos Disponibles

```
1  export async function getAvailableContacts() {
2    const supabase = await createClient()
3    const {
4      data: { user },
5    } = await supabase.auth.getUser()
6
7    if (!user) {
8      return { error: "No autenticado" }
9    }
10
11   const result = await getAllActiveUsers()
12
13   if (result.error) {
14     return { error: result.error }
15   }
16
17   // Retornar todos los usuarios activos (ya filtrados para excluir
       usuario actual)
18   return { professionals: result.users || [] }
19 }
```

Listing 60: Función getAvailableContacts completa

44.2 Archivo role-actions.ts

44.2.1 updateUserRole - Actualizar Rol de Usuario

```
1  export async function updateUserRole(
2    userId: string,
3    role: UserRole,
4    isApproved: boolean = false
5  ): Promise<{ success: boolean; error?: string }> {
6    try {
7      const supabase = await createClient()
8      const admin = await isAdmin()
9
10     if (!admin) {
```

```
      return { success: false, error: "No tienes permisos de
    administrador" }
    }

    const { error } = await supabase
      .from("user\_roles")
      .update({
        role,
        is\_approved: role === "professional" ? isApproved : true,
        updated\_at: new Date().toISOString(),
      })
      .eq("user\_id", userId)

    if (error) throw error

    return { success: true }
  } catch (error) {
    console.error("Error updating user role:", error)
    return { success: false, error: "Error al actualizar rol" }
  }
}
```

Listing 61: Función updateUserRole completa

44.3 Archivo accessibility-actions.ts

44.3.1 updateUserPreferences - Actualizar Preferencias de Accesibilidad

```
export async function updateUserPreferences(preferences: {
  color\_blind\_mode?: string
  high\_contrast?: boolean
  large\_text?: boolean
  reduce\_motion?: boolean
  screen\_reader\_optimized?: boolean
}) {
  const supabase = await createClient()
  const {
    data: { user },
  } = await supabase.auth.getUser()

  if (!user) {
    return { error: "No autenticado" }
  }

  const { error } = await supabase.from("user\_preferences").upsert(
    {
      user\_id: user.id,
      ...preferences,
    },
    {
      onConflict: "user\_id",
    },
  )

  if (error) {
    return { error: error.message }
  }
```

```
30
31   return { success: true }
32 }
```

Listing 62: Función updateUserPreferences completa

45 Componentes React

45.1 Página Principal - MessagesPage

El componente principal que coordina toda la funcionalidad del módulo de mensajería.

```
1  import { redirect } from "next/navigation"
2  import { createClient } from "@/lib/supabase/server"
3  import { MessagingInterface } from "@/components/messaging/messaging-
       interface"
4  import { Button } from "@/components/ui/button"
5  import Link from "next/link"
6  import { ArrowLeft } from "lucide-react"
7
8  export default async function MessagesPage() {
9    const supabase = await createClient()
10   const {
11     data: { user },
12   } = await supabase.auth.getUser()
13
14   if (!user) {
15     redirect("/auth/login")
16   }
17
18   return (
19     <div className="min-h-screen bg-gradient-to-br from-blue-50 to-
       indigo-100 dark:from-gray-900 dark:to-gray-800">
20       <div className="container mx-auto px-4 py-8">
21         <div className="mb-6">
22           <Button variant="outline" asChild>
23             <Link href="/">
24               <ArrowLeft className="h-4 w-4 mr-2" />
25               Volver al inicio
26             </Link>
27           </Button>
28         </div>
29
30         <div className="mb-8">
31           <h1 className="text-4xl font-bold text-gray-900 dark:text-
       white mb-2">Mensajes</h1>
32           <p className="text-lg text-gray-700 dark:text-gray-300">
       Comunicate con otros usuarios de FitTrack</p>
33         </div>
34
35         <MessagingInterface userId={user.id} />
36       </div>
37     </div>
38   )
39 }
```

Listing 63: app/messages/page.tsx - Estructura completa

Características del componente:

- **Verificación de autenticación**: Redirige a login si no está autenticado
- **Diseño responsivo**: Adaptable a diferentes tamaños de pantalla
- **Tema oscuro**: Soporte completo para modo oscuro
- **Navegación**: Botón de regreso al dashboard principal
- **Server Component**: Renderizado en servidor para mejor performance

45.2 Componente MessagingInterface

Interfaz principal de mensajería con lista de contactos y área de chat.

```
"use client"

import { useState, useEffect } from "react"
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card"
import { Button } from "@/components/ui/button"
import { Input } from "@/components/ui/input"
import { ScrollArea } from "@/components/ui/scroll-area"
import { Avatar, AvatarFallback, AvatarImage } from "@/components/ui/avatar"
import { getAvailableContacts, getMessages, sendMessage } from "@/lib/messaging-actions"
import { useToast } from "@/hooks/use-toast"
import { Send, MessageSquare, Search } from "lucide-react"

interface Professional {
  id: string
  email: string
  full\_name: string
  is\_professional?: boolean
  profile\_photo\_url?: string
}

interface Message {
  id: string
  sender\_id: string
  receiver\_id: string
  content: string
  read: boolean
  created\_at: string
}

interface MessagingInterfaceProps {
  userId: string
}

export function MessagingInterface({ userId }: MessagingInterfaceProps) {
  const [professionals, setProfessionals] = useState<Professional[]>([])
  const [selectedProfessional, setSelectedProfessional] = useState<Professional | null>(null)
  const [messages, setMessages] = useState<Message[]>([])
```

```
38  const [newMessage, setNewMessage] = useState("")
39  const [loading, setLoading] = useState(false)
40  const [searchTerm, setSearchTerm] = useState("")
41  const [userFilter, setUserFilter] = useState<"all" | "professionals" |
      "non-professionals">("all")
42  const { toast } = useToast()
43
44  useEffect(() => {
45    loadProfessionals()
46  }, [])
47
48  useEffect(() => {
49    if (selectedProfessional) {
50      loadMessages(selectedProfessional.id)
51      const interval = setInterval(() => loadMessages(
     selectedProfessional.id), 5000)
52      return () => clearInterval(interval)
53    }
54  }, [selectedProfessional])
55
56  const loadProfessionals = async () => {
57    console.log("[v0] Loading professionals...")
58    const result = await getAvailableContacts()
59    console.log("[v0] getAvailableContacts result:", result)
60
61    if (result.error) {
62      toast({
63        title: "Error",
64        description: result.error,
65        variant: "destructive",
66      })
67    } else {
68      console.log("[v0] Setting professionals:", result.professionals)
69      setProfessionals(result.professionals || [])
70    }
71  }
72
73  const loadMessages = async (professionalId: string) => {
74    const result = await getMessages(professionalId)
75    if (result.error) {
76      toast({
77        title: "Error",
78        description: result.error,
79        variant: "destructive",
80      })
81    } else {
82      setMessages(result.messages || [])
83    }
84  }
85
86  const handleSendMessage = async () => {
87    if (!selectedProfessional || !newMessage.trim()) return
88
89    setLoading(true)
90    const result = await sendMessage(selectedProfessional.id, newMessage
     )
91
92    if (result.error) {
```

```
93        toast({
94          title: "Error",
95          description: result.error,
96          variant: "destructive",
97        })
98      } else {
99        setNewMessage("")
100       loadMessages(selectedProfessional.id)
101     }
102     setLoading(false)
103   }
104
105   // Resto del componente...
106 }
```

Listing 64: components/messaging/messaging-interface.tsx - Estructura principal

Características del componente:

- **Estado centralizado**: Maneja todos los estados de la aplicación
- **Actualización automática**: Polling cada 5 segundos para nuevos mensajes
- **Filtrado y búsqueda**: Filtros por tipo de usuario y búsqueda por nombre
- **Manejo de errores**: Notificaciones toast para errores
- **TypeScript**: Tipado estricto para mayor seguridad
- **Interfaz responsiva**: Diseño adaptable a diferentes pantallas

45.3 Componente AccessibilitySettings

Configuraciones avanzadas de accesibilidad para usuarios.

```
1  "use client"
2
3  import { useState } from "react"
4  import { Card, CardContent, CardDescription, CardHeader, CardTitle }
       from "@/components/ui/card"
5  import { Label } from "@/components/ui/label"
6  import { Switch } from "@/components/ui/switch"
7  import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue }
       from "@/components/ui/select"
8  import { Button } from "@/components/ui/button"
9  import { updateUserPreferences } from "@/lib/accessibility-actions"
10 import { useToast } from "@/hooks/use-toast"
11 import { Eye, Type, Zap, Volume2, Palette, Moon, Sun } from "lucide-
       react"
12 import { useTheme } from "next-themes"
13
14 interface AccessibilitySettingsProps {
15   initialPreferences: any
16 }
17
18 export function AccessibilitySettings({ initialPreferences }:
       AccessibilitySettingsProps) {
19   const [preferences, setPreferences] = useState(initialPreferences)
```

```
20   const [loading, setLoading] = useState(false)
21   const { toast } = useToast()
22   const { theme, setTheme } = useTheme()
23
24   const handleSave = async () => {
25     setLoading(true)
26     const result = await updateUserPreferences(preferences)
27
28     if (result.error) {
29       toast({
30         title: "Error",
31         description: result.error,
32         variant: "destructive",
33       })
34     } else {
35       toast({
36         title: "Exito",
37         description: "Preferencias guardadas correctamente",
38       })
39       // Aplicar preferencias al documento
40       applyPreferences()
41     }
42     setLoading(false)
43   }
44
45   const applyPreferences = () => {
46     const root = document.documentElement
47
48     // Aplicar modo de daltonismo
49     root.setAttribute("data-color-blind-mode", preferences.color\_blind\
      _mode)
50
51     // Aplicar alto contraste
52     if (preferences.high\_contrast) {
53       root.classList.add("high-contrast")
54     } else {
55       root.classList.remove("high-contrast")
56     }
57
58     // Aplicar texto grande
59     if (preferences.large\_text) {
60       root.classList.add("large-text")
61     } else {
62       root.classList.remove("large-text")
63     }
64
65     // Aplicar reducir movimiento
66     if (preferences.reduce\_motion) {
67       root.classList.add("reduce-motion")
68     } else {
69       root.classList.remove("reduce-motion")
70     }
71   }
72
73   // Resto del componente...
74 }
```

Listing 65: components/accessibility/accessibility-settings.tsx - Estructura principal

Características del componente:

- **Configuraciones múltiples**: Modo de daltonismo, alto contraste, texto grande, etc.
- **Aplicación inmediata**: Los cambios se aplican al DOM en tiempo real
- **Persistencia**: Las preferencias se guardan en la base de datos
- **Interfaz intuitiva**: Switches y selectores fáciles de usar
- **Feedback visual**: Notificaciones de éxito/error
- **Tema integrado**: Integración con el sistema de temas

46 Sistema de Roles y Permisos

46.1 Tipos de Roles

El sistema implementa tres tipos de roles principales:

1. **Usuario (user)**: Rol por defecto para todos los usuarios registrados
2. **Profesional (professional)**: Usuarios aprobados por administradores para brindar servicios
3. **Administrador (admin)**: Usuarios con permisos completos del sistema

46.2 Jerarquía de Permisos

Funcionalidad	Usuario	Profesional	Admin
Enviar mensajes	X	X	X
Recibir mensajes	X	X	X
Ver contactos	X	X	X
Gestionar roles	-	-	X
Aprobar profesionales	-	-	X
Ver todos los usuarios	-	-	X
Configurar accesibilidad	X	X	X

Cuadro 2: Matriz de permisos por rol

46.3 Funciones de Administración

46.3.1 isAdmin - Verificar Administrador

```
export async function isAdmin() {
  try {
    const supabase = await createClient()
    const {
      data: { user },
      error: userError,
    } = await supabase.auth.getUser()

    if (userError) {
      console.error("[v0] Error getting user:", userError)
      return false
    }

    if (!user) {
      return false
    }

    const { data, error } = await supabase
      .from("user\_roles")
      .select("role")
      .eq("user\_id", user.id)
      .maybeSingle()

    if (error) {
      console.error("[v0] Error checking admin role:", error)
      return false
    }

    if (!data) {
      return false
    }

    return data.role === "admin"
  } catch (error) {
    console.error("[v0] Exception in isAdmin:", error)
    return false
  }
}
```

Listing 66: Función isAdmin completa

46.3.2 getAllUsers - Obtener Todos los Usuarios

```
export async function getAllUsers() {
  const supabase = await createClient()

  if (!(await isAdmin())) {
    return { error: "No autorizado" }
  }

  const { data: users, error } = await supabase
    .from("user\_roles")
    .select(‘
      user\_id,
      role,
      is\_active,
      is\_professional,
```

```
      created\_at,
      updated\_at,
      user:user\_id (
        id,
        email,
        raw\_user\_meta\_data
      )
    `)
    .order("created\_at", { ascending: false })

  if (error) {
    console.error("[v0] Error getting all users:", error)
    return { error: error.message }
  }

  return {
    users: users?.map((user) => ({
      id: user.user\_id,
      email: user.user?.email || "N/A",
      role: user.role,
      is\_active: user.is\_active,
      is\_professional: user.is\_professional,
      created\_at: user.created\_at,
      updated\_at: user.updated\_at,
    })) || [],
  }
}
```

Listing 67: Función getAllUsers completa

47 Configuraciones de Accesibilidad

47.1 Tipos de Configuraciones

47.1.1 Modo de Daltonismo

Soporte para diferentes tipos de daltonismo:

- **Ninguno**: Sin ajustes de color
- **Protanopía**: Dificultad con rojo-verde (ausencia de conos L)
- **Deuteranopía**: Dificultad con rojo-verde (ausencia de conos M)
- **Tritanopía**: Dificultad con azul-amarillo (ausencia de conos S)

Configuraciones adicionales: Alto contraste, texto grande, reducir movimiento y optimización para lectores de pantalla.

47.2 Implementación CSS

```
/* Modo de daltonismo - Protanopia */
[data-color-blind-mode="protanopia"] {
  --primary: #0066cc;
```

```
  --secondary: #00cc66;
  --accent: #cc6600;
}

/* Modo de daltonismo - Deuteranopia */
[data-color-blind-mode="deuteranopia"] {
  --primary: #0066cc;
  --secondary: #cc0066;
  --accent: #66cc00;
}

/* Modo de daltonismo - Tritanopia */
[data-color-blind-mode="tritanopia"] {
  --primary: #cc0066;
  --secondary: #00cc66;
  --accent: #0066cc;
}

/* Alto contraste */
.high-contrast {
  --background: #000000;
  --foreground: #ffffff;
  --primary: #ffffff;
  --secondary: #cccccc;
}

/* Texto grande */
.large-text {
  font-size: 1.2em;
}

.large-text h1 { font-size: 2.5em; }
.large-text h2 { font-size: 2em; }
.large-text h3 { font-size: 1.75em; }

/* Reducir movimiento */
.reduce-motion * {
  animation-duration: 0.01ms !important;
  animation-iteration-count: 1 !important;
  transition-duration: 0.01ms !important;
}
```

Listing 68: Implementación de estilos de accesibilidad

48 Ejemplos Prácticos de Uso

48.1 Ejemplos de Inserción a Base de Datos

Crear un Usuario con Rol:

```
-- Crear rol de usuario normal
INSERT INTO public.user\_roles (
    user\_id,
    role,
    is\_active,
    is\_professional
) VALUES (
```

```
    '123e4567-e89b-12d3-a456-426614174000',
    'user',
    true,
    false
);

-- Crear rol de profesional (requiere aprobacion)
INSERT INTO public.user\_roles (
    user\_id,
    role,
    is\_active,
    is\_professional,
    approved\_by,
    approved\_at
) VALUES (
    '456e7890-e89b-12d3-a456-426614174001',
    'professional',
    true,
    true,
    '789e0123-e89b-12d3-a456-426614174002',
    NOW()
);

-- Crear rol de administrador
INSERT INTO public.user\_roles (
    user\_id,
    role,
    is\_active,
    is\_professional
) VALUES (
    '789e0123-e89b-12d3-a456-426614174002',
    'admin',
    true,
    false
);
```

Listing 69: Ejemplo de inserción de rol de usuario

Enviar un Mensaje:

```
-- Crear conversacion
INSERT INTO public.conversations (
    user1\_id,
    user2\_id,
    last\_message\_at
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000',
    '456e7890-e89b-12d3-a456-426614174001',
    NOW()
);

-- Enviar mensaje
INSERT INTO public.messages (
    sender\_id,
    receiver\_id,
    content,
    read
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000',
    '456e7890-e89b-12d3-a456-426614174001',
```

```
    'Hola, necesito ayuda con mi rutina de ejercicios',
    false
);
```

Listing 70: Ejemplo de envío de mensaje

48.1.1 Configurar Preferencias de Accesibilidad

```
INSERT INTO public.user\_preferences (
    user\_id,
    color\_blind\_mode,
    high\_contrast,
    large\_text,
    reduce\_motion,
    screen\_reader\_optimized
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000',
    'protanopia',
    true,
    false,
    false,
    true
);
```

Listing 71: Ejemplo de configuración de accesibilidad

48.2 Consultas Útiles

48.2.1 Obtener Mensajes de una Conversación

```
SELECT
    m.id,
    m.content,
    m.created\_at,
    m.read,
    sender.email as sender\_email,
    receiver.email as receiver\_email
FROM public.messages m
JOIN auth.users sender ON m.sender\_id = sender.id
JOIN auth.users receiver ON m.receiver\_id = receiver.id
WHERE (m.sender\_id = '123e4567-e89b-12d3-a456-426614174000'
       AND m.receiver\_id = '456e7890-e89b-12d3-a456-426614174001')
   OR (m.sender\_id = '456e7890-e89b-12d3-a456-426614174001'
       AND m.receiver\_id = '123e4567-e89b-12d3-a456-426614174000')
ORDER BY m.created\_at ASC;
```

Listing 72: Consulta de mensajes de conversación

Obtener Usuarios por Rol:

```
SELECT
    u.id,
    u.email,
    ur.role,
    ur.is\_active,
    ur.is\_professional,
    ur.created\_at
```

```
FROM auth.users u
JOIN public.user\_roles ur ON u.id = ur.user\_id
WHERE ur.role = 'professional'
  AND ur.is\_active = true
  AND ur.is\_professional = true
ORDER BY ur.created\_at DESC;
```

Listing 73: Consulta de usuarios por rol

48.2.2 Obtener Estadísticas de Mensajería

```
SELECT
    COUNT(*) as total\_messages,
    COUNT(CASE WHEN read = false THEN 1 END) as unread\_messages,
    COUNT(DISTINCT sender\_id) as unique\_senders,
    COUNT(DISTINCT receiver\_id) as unique\_receivers,
    MAX(created\_at) as last\_message\_time
FROM public.messages
WHERE sender\_id = '123e4567-e89b-12d3-a456-426614174000'
   OR receiver\_id = '123e4567-e89b-12d3-a456-426614174000';
```

Listing 74: Consulta de estadísticas de mensajería

49 Flujos de Datos Completos

49.1 Flujo de Envío de Mensaje

1. **Usuario escribe mensaje** en MessagingInterface
2. **Validación en cliente** de contenido no vacío
3. **Envío a Server Action** sendMessage
4. **Verificación de autenticación** en servidor
5. **Creación/actualización de conversación** en tabla conversations
6. **Inserción del mensaje** en tabla messages
7. **Actualización de timestamp** de conversación
8. **Respuesta de éxito** al cliente
9. **Actualización de UI** con nuevo mensaje

49.2 Flujo de Carga de Contactos

1. **Inicialización del componente** MessagingInterface
2. **Llamada a getAvailableContacts** Server Action
3. **Verificación de autenticación** en servidor
4. **Consulta a getAllActiveUsers** función RPC

5. **Filtrado de usuarios** (excluir usuario actual)
6. **Formateo de datos** para presentación
7. **Retorno de lista** de contactos disponibles
8. **Renderizado en UI** con avatares y roles

49.3 Flujo de Actualización de Roles

1. **Administrador selecciona usuario** en panel de administración
2. **Selección de nuevo rol** (usuario/profesional/admin)
3. **Envío a updateUserRole** Server Action
4. **Verificación de permisos** de administrador
5. **Actualización en user_roles** tabla
6. **Registro de aprobación** (si es profesional)
7. **Respuesta de éxito** al cliente
8. **Actualización de UI** con nuevo rol

50 Características Avanzadas

50.1 Sistema de Notificaciones en Tiempo Real

El módulo implementa un sistema de polling para actualizaciones en tiempo real:

```
useEffect(() => {
  if (selectedProfessional) {
    loadMessages(selectedProfessional.id)
    const interval = setInterval(() => loadMessages(selectedProfessional
   .id), 5000)
    return () => clearInterval(interval)
  }
}, [selectedProfessional])
```

Listing 75: Implementación de polling

50.2 Filtrado y Búsqueda Avanzada

```
const filteredProfessionals = professionals.filter((prof) => {
  const matchesSearch =
    prof.full\_name.toLowerCase().includes(searchTerm.toLowerCase()) ||
    prof.email.toLowerCase().includes(searchTerm.toLowerCase())

  const matchesFilter =
    userFilter === "all" ||
    (userFilter === "professionals" && prof.is\_professional) ||
    (userFilter === "non-professionals" && !prof.is\_professional)
```

```
10
11   return matchesSearch && matchesFilter
12 })
```

Listing 76: Filtrado de contactos

50.3 Aplicación Dinámica de Preferencias

```
1  const applyPreferences = () => {
2    const root = document.documentElement
3
4    // Aplicar modo de daltonismo
5    root.setAttribute("data-color-blind-mode", preferences.color\_blind\
       _mode)
6
7    // Aplicar alto contraste
8    if (preferences.high\_contrast) {
9      root.classList.add("high-contrast")
10   } else {
11     root.classList.remove("high-contrast")
12   }
13
14   // Aplicar texto grande
15   if (preferences.large\_text) {
16     root.classList.add("large-text")
17   } else {
18     root.classList.remove("large-text")
19   }
20
21   // Aplicar reducir movimiento
22   if (preferences.reduce\_motion) {
23     root.classList.add("reduce-motion")
24   } else {
25     root.classList.remove("reduce-motion")
26   }
27 }
```

Listing 77: Aplicación de preferencias de accesibilidad

51 Mejores Prácticas de Desarrollo

51.1 Seguridad

- **Validación de entrada**: Siempre validar datos en Server Actions
- **Autenticación**: Verificar usuario en cada operación
- **RLS**: Usar Row Level Security en todas las tablas
- **Sanitización**: Limpiar inputs del usuario
- **Logs de seguridad**: Registrar operaciones sensibles
- **Verificación de roles**: Validar permisos antes de operaciones

51.2 Performance

- **Índices de base de datos**: Optimizar consultas frecuentes
- **Polling eficiente**: Intervalos apropiados para actualizaciones
- **Lazy loading**: Cargar componentes bajo demanda
- **Cache**: Usar revalidatePath para actualizar cache
- **Optimización de consultas**: Usar joins apropiados

51.3 Accesibilidad

- **ARIA labels**: Implementar etiquetas descriptivas
- **Navegación por teclado**: Soporte completo para teclado
- **Alto contraste**: Asegurar legibilidad en todos los modos
- **Lectores de pantalla**: Optimizar para tecnologías asistivas
-
- **Reducir movimiento**: Respetar preferencias de movimiento

51.4 Mantenibilidad

- **TypeScript**: Usar tipado estricto en todos los componentes
- **Interfaces**: Definir interfaces claras para props
- **Separación de responsabilidades**: Lógica en Server Actions
- **Reutilización**: Componentes modulares y reutilizables
- **Documentación**: Comentar código complejo

52 Solución de Problemas

52.1 Problemas Comunes

52.1.1 Error: Usuario no autenticado

Síntomas: Server Actions retornan error de autenticación **Causa**: Usuario no está logueado o sesión expirada **Solución**: Verificar estado de autenticación y redirigir a login

52.1.2 Error: Sin permisos de administrador

Síntomas: Error al intentar gestionar roles **Causa**: Usuario no tiene rol de administrador **Solución**: Verificar rol del usuario con isAdmin()

52.1.3 Error: Mensajes no se actualizan

Síntomas: Los mensajes no aparecen en tiempo real **Causa**: Polling deshabilitado o error en consulta **Solución**: Verificar intervalo de polling y consultas de base de datos

52.1.4 Error: Preferencias no se aplican

Síntomas: Los cambios de accesibilidad no se reflejan **Causa**: Función applyPreferences no se ejecuta **Solución**: Verificar llamada a applyPreferences() después de guardar

52.2 Debugging

52.2.1 Logs del Cliente

```
// Habilitar logs detallados
console.log('Professionals:', professionals)
console.log('Selected Professional:', selectedProfessional)
console.log('Messages:', messages)
console.log('User Filter:', userFilter)
console.log('Search Term:', searchTerm)

// Verificar estado de autenticacion
console.log('User ID:', userId)
console.log('Is Loading:', loading)
```

Listing 78: Debugging en cliente

52.2.2 Logs del Servidor

```
// En Server Actions
console.log('Action called with:', { userId, data })
console.log('User authenticated:', !!user)
console.log('User role:', userRole)

// En consultas de base de datos
console.log('Query result:', { data, error })
console.log('RLS policies active:', true)
```

Listing 79: Debugging en servidor

53 Integración con Otros Módulos

53.1 Integración con Sistema de Usuarios

El módulo de mensajería se integra estrechamente con el sistema de usuarios:

- **Perfiles de usuario**: Utiliza datos de user_profiles para mostrar información
- **Avatares**: Integra con sistema de avatares de Supabase Storage
- **Metadatos**: Utiliza raw_user_meta_data para información adicional

53.2 Integración con Panel de Administración

- **Gestión de roles**: Administradores pueden gestionar roles desde el panel
- **Aprobación de profesionales**: Sistema de aprobación para profesionales
- **Estadísticas**: Métricas de uso del sistema de mensajería

53.3 Integración con Sistema de Accesibilidad

- **Preferencias globales**: Las configuraciones se aplican a toda la aplicación
- **Persistencia**: Las preferencias se guardan por usuario
- **Aplicación dinámica**: Los cambios se aplican sin recargar la página

54 Conclusión

El módulo de mensajería de FitTrack representa una solución integral para la comunicación entre usuarios y profesionales, combinada con un sistema robusto de gestión de roles y configuraciones avanzadas de accesibilidad. Su arquitectura modular, seguridad robusta y enfoque en la inclusividad lo convierten en una herramienta poderosa para facilitar la interacción en el ecosistema FitTrack.

Características destacadas:

- Sistema de mensajería en tiempo real robusto
- Gestión de roles y permisos granular
- Configuraciones de accesibilidad completas
- Seguridad robusta con RLS
- Interfaz intuitiva y responsiva
- Integración completa con el ecosistema FitTrack
- Código bien documentado y tipado

Para contribuir al desarrollo del módulo:

1. Seguir las convenciones de TypeScript establecidas
2. Implementar tests para nuevas funcionalidades
3. Documentar cambios en políticas de seguridad
4. Mantener compatibilidad con el sistema de roles
5. Respetar las mejores prácticas de accesibilidad
6. Validar cambios con usuarios de diferentes capacidades

55 APIs y Endpoints

55.1 API Routes

55.1.1 Análisis de Comidas

`POST /api/analyze`

- Analiza imágenes de comida usando Gemini AI
- Requiere autenticación
- Retorna análisis nutricional detallado

55.1.2 Chat con IA

`POST /api/chat`

- Interfaz de chat con el asistente nutricional
- Utiliza Gemini 1.5 Flash
- Contexto personalizado del usuario

55.1.3 Estadísticas

`GET /api/stats`

- Obtiene estadísticas del usuario
- Datos agregados de entrenamientos y comidas

55.2 Server Actions

Las Server Actions se encuentran en `lib/` y manejan la lógica de negocio:

- `auth-actions.ts` - Autenticación
- `gym-actions.ts` - Operaciones de gimnasio
- `running-actions.ts` - Operaciones de running
- `health-actions.ts` - Operaciones de salud
- `messaging-actions.ts` - Sistema de mensajería
- `admin-actions.ts` - Funciones de administración

56 Configuración y Deployment

56.1 Variables de Entorno

Crear archivo `.env.local`:

```
# Google Gemini API Key
GOOGLE_GENERATIVE_AI_API_KEY=tu_clave_de_gemini_aqui

# Supabase Configuration
NEXT_PUBLIC_SUPABASE_URL=tu_supabase_url
NEXT_PUBLIC_SUPABASE_ANON_KEY=tu_supabase_anon_key
```

Listing 80: Variables de entorno requeridas

56.2 Configuración de Supabase

56.2.1 Storage Buckets

- `avatars` - Fotos de perfil de usuarios
- Configurado como público con políticas RLS

56.2.2 Políticas de Storage

```
-- Permitir acceso publico a avatars
CREATE POLICY "Avatar images are publicly accessible"
ON storage.objects FOR SELECT
USING (bucket_id = 'avatars');

-- Solo usuarios autenticados pueden subir
CREATE POLICY "Users can upload their own avatar"
ON storage.objects FOR INSERT
WITH CHECK (bucket_id = 'avatars' AND auth.uid()::text = (storage.
    foldername(name))[1]);
```

Listing 81: Políticas de storage para avatars

56.3 Scripts de Desarrollo

```
# Instalacion de dependencias
npm install

# Desarrollo local
npm run dev

# Build para produccion
npm run build

# Iniciar servidor de produccion
npm start

# Linting
npm run lint
```

Listing 82: Comandos de desarrollo

57 Guías de Desarrollo

57.1 Convenciones de Código

57.1.1 TypeScript

- Usar tipos estrictos en todas las funciones
- Definir interfaces para props de componentes
- Utilizar tipos de Supabase generados

57.1.2 Componentes React

- Usar Server Components por defecto
- Client Components solo cuando sea necesario
- Implementar Suspense para loading states
- Usar shadcn/ui como base de componentes

57.1.3 Estilos

- Usar Tailwind CSS para estilos
- Implementar modo oscuro con next-themes
- Seguir el sistema de diseño de shadcn/ui

57.2 Mejores Prácticas

57.2.1 Seguridad

- Siempre validar datos en Server Actions
- Usar RLS en todas las tablas
- Sanitizar inputs del usuario
- Implementar rate limiting en APIs

57.2.2 Performance

- Usar Server Components para reducir bundle size
- Implementar lazy loading para imágenes
- Optimizar consultas de base de datos
- Usar React.memo para componentes pesados

57.2.3 Accesibilidad

- Implementar ARIA labels
- Soporte para lectores de pantalla
- Navegación por teclado
- Alto contraste y texto grande

57.3 Testing

57.3.1 Estrategia de Testing

- Unit tests para utilidades y helpers
- Integration tests para Server Actions
- E2E tests para flujos críticos
- Testing de accesibilidad

58 Solución de Problemas

58.1 Problemas Comunes

58.1.1 Error de Hidratación

Problema: HTML renderizado en servidor no coincide con cliente **Solución**: Usar `suppressHydrationWarning` en elementos dinámicos

58.1.2 Error de API Key

Problema: "API key no configurada" **Solución**: Verificar archivo `.env.local` y reiniciar servidor

58.1.3 Error de Base de Datos

Problema: Tablas no existen **Solución**: Ejecutar scripts SQL en orden correcto

58.2 Logs y Debugging

58.2.1 Logs del Cliente

```
// Habilitar logs detallados
localStorage.setItem('debug', 'true');

// Verificar estado de autenticacion
console.log('User:', user);
console.log('Session:', session);
```

Listing 83: Debugging en cliente

58.2.2 Logs del Servidor

```
// En Server Actions
console.log('Action called with:', { userId, data });

// En API Routes
console.log('Request body:', await request.json());
```

Listing 84: Debugging en servidor

59 Recursos y Referencias

59.1 Documentación Oficial

- Next.js Documentation
- Supabase Documentation
- shadcn/ui Documentation
- Tailwind CSS Documentation

59.2 APIs Externas

- Google AI Studio
- Vercel AI SDK

59.3 Herramientas de Desarrollo

- Supabase Dashboard
- Vercel Dashboard
- Google AI Studio

60 Conclusión

Este manual proporciona una guía completa para desarrolladores que trabajen en FitTrack. La aplicación está diseñada con una arquitectura moderna, escalable y segura, utilizando las mejores prácticas de desarrollo web.

Para contribuir al proyecto:

1. Revisar este manual completamente
2. Configurar el entorno de desarrollo
3. Seguir las convenciones establecidas
4. Implementar tests apropiados
5. Documentar cambios significativos

Para soporte adicional, consultar la documentación oficial de las tecnologías utilizadas o contactar al equipo de desarrollo.