Modulo de Gimnasio

FitTrack

Documentacion Tecnica Completa

Este documento proporciona una guia tecnica exhaustiva del modulo de gimnasio de FitTrack, incluyendo arquitectura, implementacion, base de datos, componentes React, Server Actions y ejemplos practicos de uso.

Version 1.0

14 de octubre de 2025

Índice

1 Introduccion al Modulo de Gimnasio

El modulo de gimnasio es uno de los componentes mas complejos y funcionales de FitTrack. Permite a los usuarios registrar entrenamientos individuales, crear y gestionar rutinas personalizadas, hacer seguimiento de su progreso y acceder a un catalogo de ejercicios administrado.

1.1 Caracteristicas Principales

- **Registro de Entrenamientos**: Permite registrar ejercicios individuales con peso, repeticiones y series
- **Gestion de Rutinas**: Crear, editar y ejecutar rutinas personalizadas
- **Catalogo de Ejercicios**: Base de datos de ejercicios administrada por administradores
- **Historial de Progreso**: Seguimiento detallado del progreso a lo largo del tiempo
- **Metricas y Estadisticas**: Analisis de rendimiento y tendencias
- **Selector de Ejercicios**: Interfaz intuitiva para seleccionar ejercicios del catalogo
- **Imagenes de Ejercicios**: Soporte para imagenes en ejercicios personalizados

1.2 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**: Utilidades y helpers

2 Estructura de Base de Datos

2.1 Diagrama de Relaciones

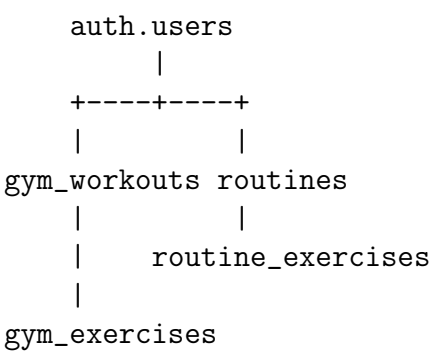

Figura 1: Diagrama de relaciones de las tablas del modulo de gimnasio

2.2 Tabla gym_workouts

Esta tabla almacena los entrenamientos individuales realizados por los usuarios.

```sql
CREATE TABLE IF NOT EXISTS public.gym_workouts (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
    exercise_name TEXT NOT NULL,
    weight_kg NUMERIC(5,2),
    repetitions INTEGER,
    sets INTEGER,
    image_url TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indices para optimizacion
CREATE INDEX IF NOT EXISTS idx_gym_workouts_user_id ON public.gym_workouts(user_id);
CREATE INDEX IF NOT EXISTS idx_gym_workouts_created_at ON public.gym_workouts(created_at);
CREATE INDEX IF NOT EXISTS idx_gym_workouts_exercise_name ON public.gym_workouts(exercise_name);
```

Listing 1: Estructura completa de $gym_workouts$

Descripcion de campos:

- `id`: Identificador unico UUID generado automaticamente
- `user_id`: Referencia al usuario propietario del entrenamiento
- `exercise_name`: Nombre del ejercicio realizado (texto libre)
- `weight_kg`: Peso utilizado en kilogramos (opcional, maximo 999.99)
- `repetitions`: Numero de repeticiones realizadas (opcional)
- `sets`: Numero de series realizadas (opcional)

- `image_url`: URL de imagen del ejercicio (opcional)
- `created_at`: Timestamp de creacion automatico

2.3 Tabla routines

Almacena las rutinas personalizadas creadas por los usuarios.

```
CREATE TABLE IF NOT EXISTS public.routines (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
    name TEXT NOT NULL,
    description TEXT,
    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
);

-- Indices para optimizacion
CREATE INDEX IF NOT EXISTS idx_routines_user_id ON public.routines(
    user_id);
CREATE INDEX IF NOT EXISTS idx_routines_created_at ON public.routines(
    created_at);
```

Listing 2: Estructura completa de routines

Descripcion de campos:

- `id`: Identificador unico UUID de la rutina
- `user_id`: Referencia al usuario propietario
- `name`: Nombre de la rutina (requerido)
- `description`: Descripcion opcional de la rutina
- `created_at`: Timestamp de creacion
- `updated_at`: Timestamp de ultima modificacion

2.4 Tabla routine_exercises

Contiene los ejercicios que pertenecen a cada rutina con su orden especifico.

```
CREATE TABLE IF NOT EXISTS public.routine_exercises (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    routine_id UUID REFERENCES public.routines(id) ON DELETE CASCADE NOT
    NULL,
    exercise_name TEXT NOT NULL,
    weight DECIMAL(6,2) NOT NULL CHECK (weight >= 0),
    repetitions INTEGER NOT NULL CHECK (repetitions > 0),
    sets INTEGER NOT NULL CHECK (sets > 0),
    image_url TEXT,
    order_index INTEGER NOT NULL DEFAULT 0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT TIMEZONE('utc'::text,
    NOW()) NOT NULL
);
```

```sql

-- Indices para optimizacion
CREATE INDEX IF NOT EXISTS idx_routine_exercises_routine_id ON public.routine_exercises(routine_id);
CREATE INDEX IF NOT EXISTS idx_routine_exercises_order ON public.routine_exercises(routine_id, order_index);
```

Listing 3: Estructura completa de routine$_e$*xercises*

Descripcion de campos:

- `id`: Identificador unico del ejercicio en la rutina
- `routine_id`: Referencia a la rutina padre
- `exercise_name`: Nombre del ejercicio
- `weight`: Peso en kilogramos (requerido, ¿= 0)
- `repetitions`: Repeticiones (requerido, ¿0)
- `sets`: Series (requerido, ¿0)
- `image_url`: URL de imagen opcional
- `order_index`: Orden del ejercicio en la rutina
- `created_at`: Timestamp de creacion

2.5 Tabla gym_exercises

Catalogo de ejercicios administrado por administradores del sistema.

```sql
CREATE TABLE IF NOT EXISTS 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 TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indices para optimizacion
CREATE INDEX IF NOT EXISTS idx_gym_exercises_category ON gym_exercises(category);
CREATE INDEX IF NOT EXISTS idx_gym_exercises_name ON gym_exercises(name);
```

Listing 4: Estructura completa de gym$_e$*xercises*

Descripcion de campos:

- `id`: Identificador unico del ejercicio
- `name`: Nombre del ejercicio

- `category`: Categoria del ejercicio (constraint de valores validos)
- `description`: Descripcion detallada del ejercicio
- `image_url`: URL de imagen del ejercicio
- `created_at`: Timestamp de creacion
- `updated_at`: Timestamp de ultima modificacion

2.6 Politicas de Seguridad (RLS)

Todas las tablas implementan Row Level Security para garantizar que los usuarios solo accedan a sus propios datos.

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

-- Politicas para gym_workouts
CREATE POLICY "Usuarios pueden ver sus propios entrenamientos"
ON public.gym_workouts
FOR SELECT USING (auth.uid() = user_id);

CREATE POLICY "Usuarios pueden insertar sus propios entrenamientos"
ON public.gym_workouts
FOR INSERT WITH CHECK (auth.uid() = user_id);

CREATE POLICY "Usuarios pueden modificar sus propios entrenamientos"
ON public.gym_workouts
FOR UPDATE USING (auth.uid() = user_id);

CREATE POLICY "Usuarios pueden eliminar sus propios entrenamientos"
ON public.gym_workouts
FOR DELETE USING (auth.uid() = user_id);
```

Listing 5: Politicas RLS para $\text{gym}_w orkouts$

3 Server Actions - Logica de Negocio

Las Server Actions manejan toda la logica de negocio del modulo de gimnasio. Estan implementadas en TypeScript con validacion robusta y manejo de errores.

3.1 Archivo gym-actions.ts

3.1.1 createWorkout - Crear Entrenamiento

```typescript
"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 6: 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

4 Componentes React

4.1 Pagina 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)
  }

```

```
39  const handleEditWorkout = (workout: Workout) => {
40    setEditingWorkout(workout)
41  }
42
43  const handleEditComplete = () => {
44    setEditingWorkout(null)
45    setRefreshTrigger((prev) => prev + 1)
46  }
47
48  const handleViewRoutine = (routineId: string, routineName: string) =>
     {
49    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
```

```
          {viewMode === "create-routine" && (
            <RoutineForm
              onRoutineCreated={handleRoutineCreated}
              onCancel={() => setViewMode("routines")}
            />
          )}

          {viewMode === "routine-detail" && selectedRoutine && (
            <RoutineDetail
              routineId={selectedRoutine.id}
              routineName={selectedRoutine.name}
              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 7: 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

5 Ejemplos Practicos de Uso

5.1 Ejemplos de Insercion a Base de Datos

5.1.1 Crear un Entrenamiento Individual

```sql
-- Insertar un entrenamiento de press de banca
INSERT INTO public.gym_workouts (
    user_id,
    exercise_name,
    weight_kg,
    repetitions,
    sets,
    image_url
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000', -- UUID del usuario
    'Press de Banca',
    80.00,
    12,
    3,
    'https://ejemplo.com/press-banca.jpg'
);

-- Insertar un entrenamiento de sentadillas
INSERT INTO public.gym_workouts (
    user_id,
    exercise_name,
    weight_kg,
    repetitions,
    sets
) VALUES (
    '123e4567-e89b-12d3-a456-426614174000',
    'Sentadillas',
    100.00,
    10,
    4
);
```

Listing 8: Ejemplo de insercion en $\text{gym}_{w}orkouts$

5.1.2 Crear una Rutina Completa

```sql
-- 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'
);

-- Obtener el ID de la rutina recien creada
-- (En la aplicacion esto se maneja automaticamente)

-- 2. Agregar ejercicios a la rutina
INSERT INTO public.routine_exercises (
    routine_id,
    exercise_name,
    weight,
```

```
    repetitions,
    sets,
    order_index
) VALUES
    ('456e7890-e89b-12d3-a456-426614174001', 'Press de Banca', 80.00,
    12, 3, 0),
    ('456e7890-e89b-12d3-a456-426614174001', 'Aperturas con Mancuernas',
    25.00, 15, 3, 1),
    ('456e7890-e89b-12d3-a456-426614174001', 'Press Frances', 30.00, 12,
    3, 2),
    ('456e7890-e89b-12d3-a456-426614174001', 'Fondos para Triceps',
    0.00, 10, 3, 3);
```

Listing 9: Ejemplo de creacion de rutina completa

5.1.3 Consultas Utiles

```
-- Obtener todos los entrenamientos de un usuario con detalles
SELECT
    gw.exercise_name,
    gw.weight_kg,
    gw.repetitions,
    gw.sets,
    gw.created_at,
    (gw.weight_kg * gw.repetitions * gw.sets) as volumen_total
FROM public.gym_workouts gw
WHERE gw.user_id = '123e4567-e89b-12d3-a456-426614174000'
ORDER BY gw.created_at DESC;

-- Obtener progreso de un ejercicio especifico
SELECT
    exercise_name,
    weight_kg,
    repetitions,
    sets,
    created_at,
    (weight_kg * repetitions * sets) as volumen
FROM public.gym_workouts
WHERE user_id = '123e4567-e89b-12d3-a456-426614174000'
  AND exercise_name = 'Press de Banca'
ORDER BY created_at ASC;

-- Obtener rutinas con conteo de ejercicios
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 10: Consultas utiles para analisis

5.2 Flujos de Datos Completos

5.2.1 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

5.2.2 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)

6 Caracteristicas Avanzadas

6.1 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

6.2 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 × reps × series)
- **Tendencias de entrenamiento**: Frecuencia y consistencia
- **Comparativas temporales**: Progreso mensual/semanal
- **Analisis por ejercicio**: Progreso especifico por ejercicio

6.3 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

7 Mejores Practicas de Desarrollo

7.1 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

7.2 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

7.3 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

8 Solucion de Problemas

8.1 Problemas Comunes

8.1.1 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`

8.1.2 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

8.1.3 Error: Politica RLS violada

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

8.2 Debugging

8.2.1 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 11: Debugging en cliente

8.2.2 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 12: Debugging en servidor

9 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