

# MÓDULO DE ADMINISTRADOR FitTrack

Manual del Programador

15 de octubre de 2025

Índice

# 1. Introducción al Módulo de Administrador

El módulo de administrador en FitTrack es un sistema completo de gestión que permite a usuarios con privilegios administrativos controlar y supervisar toda la aplicación. Este módulo está diseñado con un enfoque de seguridad robusto, utilizando Row Level Security (RLS) de Supabase y un sistema de roles bien definido.

## 1.1. Características Principales

- **Gestión de Usuarios:** Control completo sobre roles, permisos y estado de usuarios
- **Administración de Ejercicios:** Gestión del catálogo de ejercicios disponibles
- **Sistema de Roles:** Implementación de roles de usuario, profesional y administrador
- **Seguridad Avanzada:** Protección mediante RLS y funciones de seguridad
- **Interfaz Intuitiva:** Dashboard moderno con componentes React optimizados

# 2. Arquitectura del Sistema

## 2.1. Estructura de Archivos

El módulo de administrador se organiza en la siguiente estructura:

```
1  app/  
2  |-- admin/  
3  |   |-- page.tsx           # P gina principal del admin  
4  |   |-- exercises/  
5  |       |-- page.tsx      # Gest i n de ejercicios  
6  components/  
7  |-- admin/  
8  |   |-- admin-dashboard.tsx # Dashboard principal  
9  |   |-- exercise-management.tsx # Gest i n de ejercicios  
10 lib/  
11 |-- admin-actions.ts      # Acciones del servidor  
12 scripts/  
13 |-- 11-create-messaging-system.sql  
14 |-- 12-create-admin-user.sql  
15 |-- 16-fix-user-roles-rls.sql  
16 |-- 17-add-role-management-functions.sql  
17 |-- 18-create-get-all-users-function.sql  
18 |-- 19-add-is-professional-field.sql  
19 |-- 26-create-gym-exercises-table.sql  
20
```

Listing 1: Estructura del módulo de administrador

## 2.2. Base de Datos

El sistema utiliza las siguientes tablas principales:

### 2.2.1. Tabla user\_roles

```
1 CREATE TABLE user_roles (  
2   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),  
3   user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE,  
4   role TEXT NOT NULL CHECK (role IN ('user', 'admin')),  
5   is_active BOOLEAN DEFAULT true,  
6   is_professional BOOLEAN DEFAULT false,  
7   approved_by UUID REFERENCES auth.users(id),  
8   approved_at TIMESTAMPTZ,  
9   created_at TIMESTAMPTZ DEFAULT NOW(),  
10  updated_at TIMESTAMPTZ DEFAULT NOW()  
11 );  
12
```

Listing 2: Estructura de la tabla user\_roles

### 2.2.2. Tabla gym\_exercises

```
1 CREATE TABLE gym_exercises (  
2   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),  
3   name TEXT NOT NULL,  
4   category TEXT NOT NULL CHECK (category IN ('Pecho', 'B\'iceps', 'Tr\  
'iceps', 'Hombros', 'Pierna', 'Espalda', 'Otros')),  
5   description TEXT,  
6   image_url TEXT,  
7   created_at TIMESTAMPTZ DEFAULT NOW(),  
8   updated_at TIMESTAMPTZ DEFAULT NOW()  
9 );  
10
```

Listing 3: Estructura de la tabla gym\_exercises

## 3. Sistema de Autenticación y Autorización

### 3.1. Verificación de Privilegios de Administrador

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

```
1 export async function isAdmin() {  
2   try {  
3     const supabase = await createClient()  
4     const { data: { user }, error: userError } = await supabase.auth  
5     .getUser()  
6  
7     if (userError || !user) {  
8       return false  
9     }  
10  
11     const { data, error } = await supabase  
12     .from("user_roles")  
13     .select("role")  
14     .eq("user_id", user.id)  
15     .maybeSingle()
```

```
16     if (error || !data) {
17         return false
18     }
19
20     return data.role === "admin"
21 } catch (error) {
22     console.error("[v0] Exception in isAdmin:", error)
23     return false
24 }
25 }
26
```

Listing 4: Función isAdmin en admin-actions.ts

## 3.2. Protección de Rutas

Cada página de administración implementa protección de acceso:

```
1 export default async function AdminPage() {
2     const admin = await isAdmin()
3
4     if (!admin) {
5         redirect("/")
6     }
7     // ... resto del componente
8 }
9
```

Listing 5: Protección de rutas en page.tsx

# 4. Funcionalidades del Dashboard Principal

## 4.1. 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

## 4.2. Estados y Gestión de Datos

El dashboard utiliza React hooks para el manejo de estado:

```
1  const [users, setUsers] = useState(initialUsers)
2  const [loading, setLoading] = useState<string | null>(null)
3  const [searchQuery, setSearchQuery] = useState("")
4  const { toast } = useToast()
5
```

Listing 6: Estados del componente AdminDashboard

## 4.3. Funciones de Gestión de Usuarios

### 4.3.1. Cambio de Roles

```
1  const handleRoleChange = async (userId: string, role: string) => {
2    setLoading(userId)
3    const user = users.find((u) => u.id === userId)
4    if (!user) return
5
6    const result = await updateUserRole(userId, role, user.is_active,
7    user.is_professional)
8
9    if (result.error) {
10     toast({
11       title: "Error",
12       description: result.error,
13       variant: "destructive",
14     })
15   } else {
16     toast({
17       title: " xito ",
18       description: "Rol actualizado correctamente",
19     })
20     setUsers(users.map((u) => (u.id === userId ? { ...u, role } : u)
21   ))
22   }
23   setLoading(null)
24 }
```

Listing 7: Función handleRoleChange

## 5. Gestión de Ejercicios

### 5.1. 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

## 5.2. 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

## 5.3. Función de Subida de Imágenes

```
1 export async function uploadExerciseImage(formData: FormData) {  
2   const file = formData.get("file") as File  
3  
4   if (!file) {  
5     return { error: "No se proporcion ning n archivo" }  
6   }  
7  
8   if (!file.type.startsWith("image/")) {  
9     return { error: "El archivo debe ser una imagen" }  
}
```

```

10     }
11
12     if (file.size > 5 * 1024 * 1024) {
13         return { error: "El archivo no debe superar los 5MB" }
14     }
15
16     try {
17         const supabase = await createClient()
18         const { data: { user }, error: userError } = await supabase.auth
19         .getUser()
20
21         if (userError || !user) {
22             return { error: "Usuario no autenticado" }
23         }
24
25         const fileExt = file.name.split(".").pop()
26         const fileName = `${Date.now()}-${Math.random().toString(36).
27         substring(7)}.${fileExt}`
28         const filePath = `exercise-images/${fileName}`
29
30         const { data, error: uploadError } = await supabase.storage
31         .from("exercise-images")
32         .upload(filePath, file, {
33             cacheControl: "3600",
34             upsert: false,
35         })
36
37         if (uploadError) {
38             return { error: "Error al subir la imagen" }
39         }
40
41         const { data: { publicUrl } } = supabase.storage
42         .from("exercise-images")
43         .getPublicUrl(filePath)
44
45         return { success: true, imageUrl: publicUrl }
46     } catch (error) {
47         return { error: "Error de conexión con la base de datos" }
48     }

```

Listing 8: Función uploadExerciseImage

## 6. Funciones de Base de Datos

### 6.1. Función get\_all\_users\_with\_roles

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

```

1  CREATE OR REPLACE FUNCTION get_all_users_with_roles()
2  RETURNS TABLE (
3      id uuid,
4      email text,
5      full_name text,
6      role text,

```

```

7      is_active boolean,
8      is_professional boolean,
9      created_at timestampz
10     )
11     SECURITY DEFINER
12     SET search_path = public
13     LANGUAGE plpgsql
14     AS $$
15     BEGIN
16         -- Check if the current user is an admin
17         IF NOT EXISTS (
18             SELECT 1 FROM user_roles ur_check
19             WHERE ur_check.user_id = auth.uid()
20             AND ur_check.role = 'admin'
21         ) THEN
22             RAISE EXCEPTION 'No autorizado';
23         END IF;
24
25         -- Return all users with their roles
26         RETURN QUERY
27         SELECT
28             au.id,
29             au.email::text,
30             COALESCE(
31                 au.raw_user_meta_data->>'full_name',
32                 CONCAT(
33                     COALESCE(au.raw_user_meta_data->>'first_name', ''),
34                     ' ',
35                     COALESCE(au.raw_user_meta_data->>'last_name', '')
36                 ),
37                 'Sin nombre'
38             )::text,
39             COALESCE(ur.role, 'user')::text,
40             COALESCE(ur.is_active, true),
41             COALESCE(ur.is_professional, false),
42             au.created_at
43         FROM auth.users au
44         LEFT JOIN user_roles ur ON ur.user_id = au.id
45         ORDER BY au.created_at DESC;
46     END;
47     $$;
48

```

Listing 9: Función get\_all\_users\_with\_roles

## 6.2. Función is\_admin

Función auxiliar para verificar privilegios administrativos:

```

1      CREATE OR REPLACE FUNCTION public.is_admin(check_user_id UUID)
2      RETURNS BOOLEAN AS $$
3      DECLARE
4          user_role TEXT;
5      BEGIN
6          SELECT role INTO user_role
7          FROM public.user_roles
8          WHERE user_id = check_user_id;
9

```



```
10 RETURN user_role = 'admin';
11 END;
12 $$ LANGUAGE plpgsql SECURITY DEFINER;
13
```

Listing 10: Función is\_admin

## 7. Seguridad y Row Level Security (RLS)

### 7.1. Políticas de Seguridad

El sistema implementa múltiples capas de seguridad:

#### 1. RLS en user\_roles:

```
1  -- Admins can view all roles
2  CREATE POLICY "Admins can view all roles" ON public.
   user_roles
3  FOR SELECT USING (public.is_admin(auth.uid()));
4
5  -- Admins can update roles
6  CREATE POLICY "Admins can update roles" ON public.user_roles
7  FOR UPDATE USING (public.is_admin(auth.uid()));
8
9  -- Admins can insert roles
10 CREATE POLICY "Admins can insert roles" ON public.user_roles
11 FOR INSERT WITH CHECK (public.is_admin(auth.uid()));
12
```

Listing 11: Políticas RLS para user\_roles

#### 2. RLS en gym\_exercises:

```
1  -- Anyone can view gym exercises
2  CREATE POLICY "Anyone can view gym exercises"
3  ON gym_exercises
4  FOR SELECT
5  TO authenticated
6  USING (true);
7
8  -- Only admins can manage gym exercises
9  CREATE POLICY "Only admins can manage gym exercises"
10 ON gym_exercises
11 FOR ALL
12 TO authenticated
13 USING (
14 EXISTS (
15 SELECT 1 FROM user_roles
16 WHERE user_roles.user_id = auth.uid()
17 AND user_roles.role = 'admin'
18 )
19 );
20
```

Listing 12: Políticas RLS para gym\_exercises

## 7.2. 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.

## 8. Scripts de Configuración

### 8.1. Creación del Usuario Administrador

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

```

1  -- Insert admin role for the admin user
2  -- First, create user through Supabase Auth with email: juan@ejemplo
   .com and password: 123456
3  -- Then run this script to assign the admin role
4
5  INSERT INTO public.user_roles (user_id, role, is_active, approved_at
   )
6  SELECT id, 'admin', true, NOW()
7  FROM auth.users
8  WHERE email = 'juan@ejemplo.com'
9  ON CONFLICT (user_id) DO UPDATE
10 SET role = 'admin', is_active = true, approved_at = NOW();
11
12 -- Verify the admin was created
13 SELECT u.id, u.email, ur.role, ur.is_active
14 FROM auth.users u
15 LEFT JOIN public.user_roles ur ON u.id = ur.user_id
16 WHERE u.email = 'juan@ejemplo.com';
17

```

Listing 13: Script para crear usuario administrador

### 8.2. Inserción de Ejercicios por Defecto

El sistema incluye ejercicios predefinidos para todas las categorías:

```

1  INSERT INTO gym_exercises (name, category, description) VALUES
2  -- Pecho
3  ('Press de Banca', 'Pecho', 'Ejercicio b sico para pecho con barra'
   ),
4  ('Press Inclinado', 'Pecho', 'Press de banca en banco inclinado'),
5  ('Aperturas con Mancuernas', 'Pecho', 'Aperturas para pecho'),
6  ('Fondos en Paralelas', 'Pecho', 'Fondos para pecho y tr ceps'),
7
8  -- B\iceps
9  ('Curl con Barra', 'B\iceps', 'Curl de b ceps con barra recta'),
10 ('Curl con Mancuernas', 'B\iceps', 'Curl alternado con mancuernas')
   ,
11 ('Curl Martillo', 'B\iceps', 'Curl con agarre neutro'),
12 ('Curl en Banco Scott', 'B\iceps', 'Curl concentrado en banco'),
13
14 -- Tr\iceps
15 ('Press Franc s', 'Tr\iceps', 'Extensi n de tr ceps acostado'),
16 ('Fondos para Tr\iceps', 'Tr\iceps', 'Fondos en banco'),

```

```

17 ('Extensi n en Polea', 'Tr\'iceps', 'Extensi n de tr ceps en
18 polea alta'),
19
20 -- Hombros
21 ('Press Militar', 'Hombros', 'Press de hombros con barra'),
22 ('Elevaciones Laterales', 'Hombros', 'Elevaciones laterales con
23 mancuernas'),
24 ('Elevaciones Frontales', 'Hombros', 'Elevaciones frontales'),
25 ('P jaros', 'Hombros', 'Elevaciones posteriores'),
26
27 -- Pierna
28 ('Sentadilla', 'Pierna', 'Sentadilla con barra'),
29 ('Prensa de Pierna', 'Pierna', 'Press de piernas en m quina'),
30 ('Peso Muerto', 'Pierna', 'Peso muerto convencional'),
31 ('Zancadas', 'Pierna', 'Zancadas con mancuernas'),
32 ('Extensi n de Cu driceps', 'Pierna', 'Extensi n en m quina'),
33 ('Curl Femoral', 'Pierna', 'Curl de piernas acostado'),
34 ('Elevaci n de Gemelos', 'Pierna', 'Elevaci n de pantorrillas'),
35
36 -- Espalda
37 ('Dominadas', 'Espalda', 'Dominadas con peso corporal'),
38 ('Remo con Barra', 'Espalda', 'Remo inclinado con barra'),
39 ('Remo con Mancuerna', 'Espalda', 'Remo a una mano'),
40 ('Jal n al Pecho', 'Espalda', 'Jal n en polea alta'),
41 ('Peso Muerto Rumano', 'Espalda', 'Peso muerto para espalda baja'),
42
43 -- Otros
44 ('Plancha', 'Otros', 'Plancha abdominal'),
45 ('Abdominales', 'Otros', 'Crunch abdominal'),
46 ('Cardio', 'Otros', 'Ejercicio cardiovascular');

```

Listing 14: Ejercicios por defecto

## 9. Integración con Otros Módulos

### 9.1. 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

### 9.2. 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

## 10. Consideraciones de Rendimiento

### 10.1. Índices de Base de Datos

El sistema incluye índices optimizados:

```
1  -- ndice para consultas de categoria de ejercicios
2  CREATE INDEX idx_gym_exercises_category ON gym_exercises(category);
3
4  -- ndice para consultas de usuarios profesionales
5  CREATE INDEX idx_user_roles_is_professional
6  ON user_roles(is_professional)
7  WHERE is_professional = true;
8
```

Listing 15: Índices para optimización

### 10.2. 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

## 11. Testing y Debugging

### 11.1. Logs de Debugging

El sistema incluye logging extensivo para debugging:

```
1  console.log("[v0] Form data before submission:", formData)
2  console.log("[v0] Image URL value:", formData.image_url)
3  console.log("[v0] Server action result:", result)
4
```

Listing 16: Ejemplo de logging

### 11.2. 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

## 12. Configuración de Desarrollo

### 12.1. Variables de Entorno

El módulo requiere las siguientes variables de entorno:

```
1 NEXT_PUBLIC_SUPABASE_URL=your_supabase_url
2 NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key
3 SUPABASE_SERVICE_ROLE_KEY=your_service_role_key
4
```

Listing 17: Variables de entorno necesarias

### 12.2. Dependencias

Las principales dependencias del módulo incluyen:

```
1 {
2   "@supabase/supabase-js": "^2.38.0",
3   "@radix-ui/react-select": "^2.0.0",
4   "@radix-ui/react-switch": "^1.0.3",
5   "@radix-ui/react-tabs": "^1.0.4",
6   "lucide-react": "^0.292.0",
7   "next": "14.0.0",
8   "react": "^18.0.0",
9   "tailwindcss": "^3.3.0"
10 }
11
```

Listing 18: Dependencias principales

## 13. Despliegue y Producción

### 13.1. 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

### 13.2. Monitoreo

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

## 14. 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.