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:

```
app/
        admin/
              page.tsx                  # P gina principal del admin
              exercises/
                  page.tsx             # Gesti n de ejercicios
components/
        admin/
              admin-dashboard.tsx      # Dashboard principal
              exercise-management.tsx # Gesti n de ejercicios
lib/
        admin-actions.ts             # Acciones del servidor
scripts/
        11-create-messaging-system.sql
        12-create-admin-user.sql
        16-fix-user-roles-rls.sql
        17-add-role-management-functions.sql
        18-create-get-all-users-function.sql
        19-add-is-professional-field.sql
        26-create-gym-exercises-table.sql

```

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

```
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 2: Estructura de la tabla user_roles

2.2.2. 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', 'B ceps', '
Tr ceps', 'Hombros', 'Pierna', 'Espalda', 'Otros')),
description TEXT,
image_url TEXT,
created_at TIMESTAMPTZ DEFAULT NOW(),
updated_at TIMESTAMPTZ DEFAULT NOW()
);

```

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:

```
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 4: Función isAdmin en admin-actions.ts

3.2. 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 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:

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

```

Listing 6: Estados del componente AdminDashboard

4.3. Funciones de Gestión de Usuarios

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

```

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

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

  if (!file) {
    return { error: "No se proporcion  ning n 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 conexi n con la base de datos" }
    }
  }

```

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:

```
  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 9: Función get_all_users_with_roles

6.2. 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 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**:

```
    -- 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 11: 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 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:

```
-- 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 13: Script para crear usuario administrador

8.2. 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 b sico 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 tr ceps'),

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

-- Tr ceps
('Press Franc s', 'Tr ceps', 'Extensi n de tr ceps acostado'),
('Fondos para Tr ceps', 'Tr ceps', 'Fondos en banco'),
```

```
    ('Extensi n en Polea', 'Tr ceps', 'Extensi n de tr ceps en polea alta'),
    ('Patada de Tr ceps', 'Tr ceps', 'Extensi n con mancuerna'),

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

    -- Pierna
    ('Sentadilla', 'Pierna', 'Sentadilla con barra'),
    ('Prensa de Pierna', 'Pierna', 'Press de piernas en m quina'),
    ('Peso Muerto', 'Pierna', 'Peso muerto convencional'),
    ('Zancadas', 'Pierna', 'Zancadas con mancuernas'),
    ('Extensi n de Cu driceps', 'Pierna', 'Extensi n en m quina'),
    ('Curl Femoral', 'Pierna', 'Curl de piernas acostado'),
    ('Elevaci n de Gemelos', 'Pierna', 'Elevaci n 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'),
    ('Jal n al Pecho', 'Espalda', 'Jal n 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 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:

```
-- Índice para consultas de categoría de ejercicios
CREATE INDEX idx_gym_exercises_category ON gym_exercises(category);

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

```

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:

```
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 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:

```
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 17: Variables de entorno necesarias

12.2. 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 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.