Módulo de Mensajería

FitTrack

Sistema de Comunicación y Accesibilidad

Este módulo proporciona un sistema completo de mensajería entre usuarios y profesionales, gestión de roles y permisos, y configuraciones avanzadas de accesibilidad para hacer la aplicación inclusiva para todos los usuarios.

Versión 1.0

14 de octubre de 2025

Índice

1 Introducción al 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.

1.1 Características Principales

- **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

1.2 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

2 Estructura de Base de Datos

2.1 Diagrama de Relaciones

2.2 Tabla user_roles

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

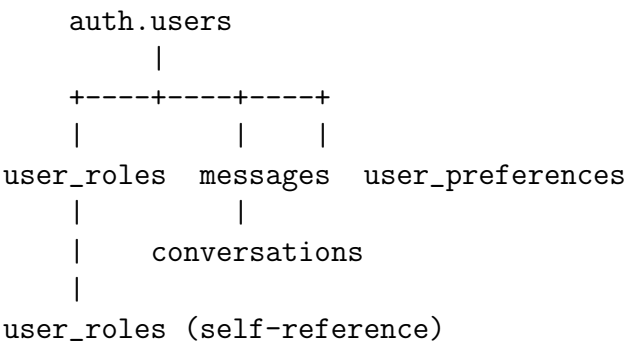

Figura 1: Diagrama de relaciones de las tablas del módulo de mensajería

```sql
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 1: 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

2.3 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 2: 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

2.4 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 3: 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

2.5 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 4: 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

2.6 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 5: 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 6: Políticas RLS para messages

3 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.

3.1 Archivo messaging-actions.ts

3.1.1 sendMessage - Enviar Mensaje

```
"use server"

import { createClient } from "@/lib/supabase/server"

export async function sendMessage(receiverId: string, content: string) {
  const supabase = await createClient()
  const {
    data: { user },
  } = await supabase.auth.getUser()

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

```
14
15    // Crear o actualizar conversacion
16    const { error: convError } = await supabase.from("conversations").
       upsert(
17      {
18        user1\_id: user.id < receiverId ? user.id : receiverId,
19        user2\_id: user.id < receiverId ? receiverId : user.id,
20        last\_message\_at: new Date().toISOString(),
21      },
22      {
23        onConflict: "user1\_id,user2\_id",
24      },
25    )
26
27    if (convError) {
28      console.error("Error creating conversation:", convError)
29    }
30
31    // Enviar mensaje
32    const { error } = await supabase.from("messages").insert({
33      sender\_id: user.id,
34      receiver\_id: receiverId,
35      content,
36    })
37
38    if (error) {
39      return { error: error.message }
40    }
41
42    return { success: true }
43  }
```

Listing 7: 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

3.1.2 getMessages - Obtener Mensajes

```
1  export async function getMessages(otherUserId: string) {
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
```

```
  const { data, error } = await supabase
    .from("messages")
    .select("*")
    .or(
      `and(sender\_id.eq.${user.id},receiver\_id.eq.${otherUserId}),and(sender\_id.eq.${otherUserId},receiver\_id.eq.${user.id})`,
    )
    .order("created\_at", { ascending: true })

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

  return { messages: data }
}
```

Listing 8: Función getMessages completa

3.1.3 getAvailableContacts - Obtener Contactos Disponibles

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

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

  const result = await getAllActiveUsers()

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

  // Retornar todos los usuarios activos (ya filtrados para excluir usuario actual)
  return { professionals: result.users || [] }
}
```

Listing 9: Función getAvailableContacts completa

3.2 Archivo role-actions.ts

3.2.1 updateUserRole - Actualizar Rol de Usuario

```
export async function updateUserRole(
  userId: string,
  role: UserRole,
  isApproved: boolean = false
): Promise<{ success: boolean; error?: string }> {
  try {
    const supabase = await createClient()
    const admin = await isAdmin()

```

```
    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 10: Función updateUserRole completa

3.3 Archivo accessibility-actions.ts

3.3.1 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 }
```

```
  }

  return { success: true }
}
```

Listing 11: Función updateUserPreferences completa

4 Componentes React

4.1 Página Principal - MessagesPage

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

```
import { redirect } from "next/navigation"
import { createClient } from "@/lib/supabase/server"
import { MessagingInterface } from "@/components/messaging/messaging-interface"
import { Button } from "@/components/ui/button"
import Link from "next/link"
import { ArrowLeft } from "lucide-react"

export default async function MessagesPage() {
  const supabase = await createClient()
  const {
    data: { user },
  } = await supabase.auth.getUser()

  if (!user) {
    redirect("/auth/login")
  }

  return (
    <div className="min-h-screen bg-gradient-to-br from-blue-50 to-indigo-100 dark:from-gray-900 dark:to-gray-800">
      <div className="container mx-auto px-4 py-8">
        <div className="mb-6">
          <Button variant="outline" asChild>
            <Link href="/">
              <ArrowLeft className="h-4 w-4 mr-2" />
              Volver al inicio
            </Link>
          </Button>
        </div>

        <div className="mb-8">
          <h1 className="text-4xl font-bold text-gray-900 dark:text-white mb-2">Mensajes</h1>
          <p className="text-lg text-gray-700 dark:text-gray-300">Comunicate con otros usuarios de FitTrack</p>
        </div>

        <MessagingInterface userId={user.id} />
      </div>
    </div>
  )
```

```
}
```

Listing 12: 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

4.2 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[]>([])
```

```
36  const [selectedProfessional, setSelectedProfessional] = useState<
     Professional | null>(null)
37  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 13: 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

4.3 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
```

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

Listing 14: 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

5 Sistema de Roles y Permisos

5.1 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

5.2 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 1: Matriz de permisos por rol

5.3 Funciones de Administración

5.3.1 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 15: Función isAdmin completa

5.3.2 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 16: Función getAllUsers completa

6 Configuraciones de Accesibilidad

6.1 Tipos de Configuraciones

6.1.1 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)

6.1.2 Alto Contraste

Mejora la legibilidad aumentando el contraste entre texto y fondo.

6.1.3 Texto Grande

Aumenta el tamaño de fuente en toda la aplicación para usuarios con dificultades visuales.

6.1.4 Reducir Movimiento

Minimiza animaciones y transiciones para usuarios sensibles al movimiento.

6.1.5 Optimización para Lectores de Pantalla

Mejora la experiencia con tecnologías asistivas como lectores de pantalla.

6.2 Implementación CSS

```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 17: Implementación de estilos de accesibilidad

7 Ejemplos Prácticos de Uso

7.1 Ejemplos de Inserción a Base de Datos

7.1.1 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 18: Ejemplo de inserción de rol de usuario

7.1.2 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 19: Ejemplo de envío de mensaje

7.1.3 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 20: Ejemplo de configuración de accesibilidad

7.2 Consultas Útiles

7.2.1 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 21: Consulta de mensajes de conversación

7.2.2 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 22: Consulta de usuarios por rol

7.2.3 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 23: Consulta de estadísticas de mensajería

8 Flujos de Datos Completos

8.1 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

8.2 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

8.3 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

9 Características Avanzadas

9.1 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 24: Implementación de polling

9.2 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)

  return matchesSearch && matchesFilter
})
```

Listing 25: Filtrado de contactos

9.3 Aplicación Dinámica de Preferencias

```
const applyPreferences = () => {
  const root = document.documentElement

  // Aplicar modo de daltonismo
  root.setAttribute("data-color-blind-mode", preferences.color\_blind\
    _mode)

  // Aplicar alto contraste
  if (preferences.high\_contrast) {
    root.classList.add("high-contrast")
  } else {
    root.classList.remove("high-contrast")
  }

  // Aplicar texto grande
  if (preferences.large\_text) {
    root.classList.add("large-text")
  } else {
    root.classList.remove("large-text")
```

```
  }

  // Aplicar reducir movimiento
  if (preferences.reduce\_motion) {
    root.classList.add("reduce-motion")
  } else {
    root.classList.remove("reduce-motion")
  }
}
```

Listing 26: Aplicación de preferencias de accesibilidad

10 Mejores Prácticas de Desarrollo

10.1 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

10.2 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

10.3 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

10.4 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

11 Solución de Problemas

11.1 Problemas Comunes

11.1.1 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

11.1.2 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()

11.1.3 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

11.1.4 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

11.2 Debugging

11.2.1 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 27: Debugging en cliente

11.2.2 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 28: Debugging en servidor

12 Integración con Otros Módulos

12.1 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

12.2 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

12.3 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

13 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