

DIAGNÓSTICO COMPLETO: Error “User not found” en CitaPlanner

Fecha: 8 de octubre de 2025

Repositorio: qhosting/citaplanner

Rama: main (commit ba4c75b)

Módulo afectado: Clientes (CRM) - Perfiles de Cliente

RESUMEN EJECUTIVO

Se ha identificado un **error crítico de arquitectura** en el módulo de gestión de clientes que impide la creación de perfiles de cliente. El error “User not found” se produce porque el sistema intenta validar la existencia de un usuario en la tabla `User` antes de crear un `ClientProfile`, pero el flujo de la aplicación **no proporciona un mecanismo para crear usuarios regulares** antes de crear sus perfiles.

Severidad: ● CRÍTICA

Impacto: Bloquea completamente la funcionalidad de creación de clientes

Módulos afectados: 4 archivos principales + potencialmente otros módulos

CAUSA RAÍZ IDENTIFICADA

Problema Principal: Arquitectura de Dos Modelos Desconectados

CitaPlanner tiene **dos modelos separados** para representar clientes:

1. **Client** (tabla `clients`): Modelo básico de cliente con datos mínimos
 - Campos: `firstName`, `lastName`, `email`, `phone`, `address`, `birthday`, `notes`
 - Relación: Pertenece a un `Tenant`
 - Usado en: Citas, Pagos, Ventas
2. **ClientProfile** (tabla `ClientProfile`): Perfil extendido de cliente (Fase 2 - CRM)
 - Relación: 1-to-1 con `User` (campo `userId` UNIQUE y REQUERIDO)
 - Campos adicionales: `gender`, `occupation`, `company`, `emergencyContact`, etc.
 - Usado en: Módulo CRM avanzado

El Conflicto

El código actual en `/app/dashboard/clients/new/page.tsx` intenta crear un `ClientProfile` directamente, pero:

```
// Línea 23-25 en new/page.tsx
if (!data.userId && session?.user?.id) {
  data.userId = session.user.id; // ✗ Usa el ID del usuario AUTENTICADO (staff)
}
```

Problema: El `userId` que se envía es el del **usuario autenticado** (profesional/admin que está creando el cliente), NO el ID de un usuario que represente al cliente.

Luego, en `clientManager.ts` línea 48-54:

```
// Valida que el userId exista en la tabla User
const user = await prisma.user.findUnique({
  where: { id: data.userId },
});

if (!user) {
  throw new Error('User not found'); // ✗ Error aquí
}
```

¿Por qué falla?

1. El sistema **NO tiene un endpoint público** para crear usuarios regulares (clientes)
2. El único endpoint de creación de usuarios es `/api/admin/master/create-user` que requiere permisos de administrador
3. La interfaz de “Nuevo Cliente” intenta crear un `ClientProfile` sin haber creado primero un `User` para ese cliente
4. El `session.user.id` que se usa es del staff, no del cliente

ARCHIVOS AFECTADOS

1. Archivo Principal del Error

`/app/lib/clients/clientManager.ts` (Línea 48-54)

```
export async function createClientProfile(data: CreateClientProfileInput) {
  try {
    // ✗ PROBLEMA: Valida existencia de User antes de crear ClientProfile
    const user = await prisma.user.findUnique({
      where: { id: data.userId },
    });

    if (!user) {
      throw new Error('User not found'); // ● ERROR AQUÍ
    }
    // ...
  }
}
```

Impacto: Bloquea completamente la creación de perfiles de cliente.

2. Frontend que Invoca el Error

`/app/app/dashboard/clients/new/page.tsx` (Línea 23-25)

```
// ❌ PROBLEMA: Usa el userId del usuario autenticado (staff)
if (!data.userId && session?.user?.id) {
  data.userId = session.user.id; // Esto es incorrecto
}
```

Impacto: Envía el userId incorrecto al backend.

3. Otros Archivos con el Mismo Patrón

a) /app/lib/clients/historyService.ts (Línea 38-45)

```
export async function getClientHistory(targetUserId: string, ...) {
  const user = await prisma.user.findUnique({
    where: { id: targetUserId },
    select: { tenantId: true, email: true, phone: true },
  });

  if (!user) {
    throw new Error('User not found'); // 🟡 MISMO PATRÓN
  }
}
```

Impacto: Falla al intentar obtener historial de un cliente si no tiene User asociado.

b) /app/lib/clients/historyService.ts (Línea 160-167)

```
export async function addHistoryEntry(data: AddHistoryEntryInput) {
  const user = await prisma.user.findUnique({
    where: { id: data.userId },
  });

  if (!user) {
    throw new Error('User not found'); // 🟡 MISMO PATRÓN
  }
}
```

Impacto: Falla al agregar entradas de historial.

c) /app/lib/clients/noteManager.ts (Línea 44-50)

```
export async function createClientNote(data: CreateClientNoteInput) {
  // Verify user exists
  const user = await prisma.user.findUnique({
    where: { id: data.createdByUserId },
  });

  if (!user) {
    throw new Error('User not found'); // 🟡 MISMO PATRÓN
  }
}
```

Impacto: Falla al crear notas de cliente (aunque aquí `createdByUserId` sí debería ser un usuario staff válido).

ANÁLISIS DE OTROS MÓDULOS

Módulos Revisados (SIN este patrón problemático):

- ✓ **Productos** (`/app/lib/services/productService.ts`)
 - No valida usuarios, solo busca productos por ID
 - No tiene el patrón "User not found"
- ✓ **Inventario** (`/app/lib/services/inventoryService.ts`)
 - No valida usuarios
 - Trabaja directamente con productos y movimientos
- ✓ **Ventas** (`/app/lib/services/saleService.ts`)
 - No valida usuarios directamente
 - Busca productos y clientes, pero no usuarios
- ✓ **Servicios** (`/app/lib/services/serviceManager.ts`)
 - No valida usuarios
 - Trabaja con servicios y categorías
- ✓ **Notificaciones** (`/app/lib/notifications/*.ts`)
 - No valida usuarios
 - Envía notificaciones sin validar existencia de User

ARQUITECTURA ACTUAL vs. ESPERADA

Arquitectura Actual (PROBLEMÁTICA)

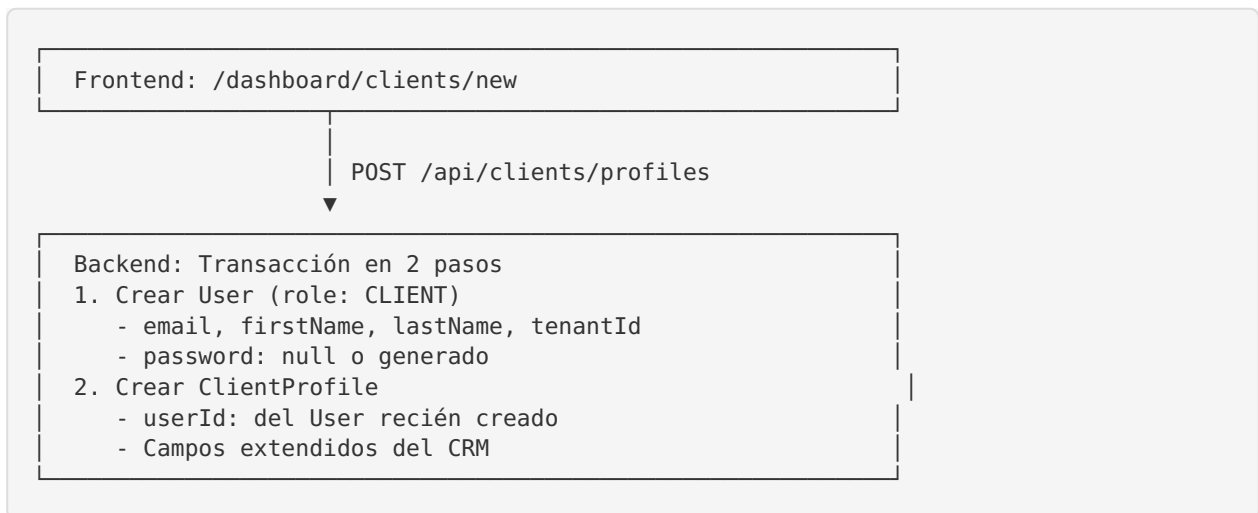


Arquitectura Esperada (CORRECTA)

Opción A: Modelo Unificado (Recomendado)



Opción B: Crear User + ClientProfile (Complejo)



PROPUESTA DE SOLUCIÓN

Solución Recomendada: Opción A - Usar modelo **Client** existente

Justificación:

1. El modelo **Client** ya existe y está integrado con Citas, Ventas y Pagos
2. No requiere crear usuarios para cada cliente
3. Más simple y directo
4. **ClientProfile** puede ser opcional para casos avanzados de CRM

Cambios Necesarios:

1. Modificar Frontend (/app/dashboard/clients/new/page.tsx)

```
// Cambiar endpoint de /api/clients/profiles a /api/clients
const response = await fetch('/api/clients', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    firstName: data.firstName,
    lastName: data.lastName,
    phone: data.phone,
    email: data.email,
    address: data.address,
    birthday: data.dateOfBirth,
    notes: data.notes,
    // NO enviar userId
  }),
});
```

2. Crear/Actualizar endpoint /api/clients/route.ts

```
export async function POST(request: NextRequest) {
  const session = await getServerSession(authOptions);
  if (!session?.user) {
    return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
  }

  const body = await request.json();

  // Crear Client (no ClientProfile)
  const client = await prisma.client.create({
    data: {
      firstName: body.firstName,
      lastName: body.lastName,
      phone: body.phone,
      email: body.email,
      address: body.address,
      birthday: body.birthday,
      notes: body.notes,
      tenantId: session.user.tenantId,
    },
  });

  return NextResponse.json({ success: true, data: client });
}
```

3. Actualizar listado de clientes

- Cambiar de /api/clients/profiles a /api/clients
- Adaptar la UI para mostrar datos del modelo Client

Solución Alternativa: Opción B - Crear User automáticamente

Solo si se requiere mantener ClientProfile:

1. Modificar `clientManager.createClientProfile()`

```
export async function createClientProfile(data: CreateClientProfileInput) {
  try {
    // Si no se proporciona userId, crear un User automáticamente
    let userId = data.userId;

    if (!userId) {
      // Crear User para el cliente
      const newUser = await prisma.user.create({
        data: {
          email: data.email || `cliente_${Date.now()}@temp.com`,
          firstName: data.firstName || 'Cliente',
          lastName: data.lastName || 'Nuevo',
          phone: data.phone,
          role: 'CLIENT',
          tenantId: session.user.tenantId, // Del usuario autenticado
          password: null, // Sin contraseña inicialmente
        },
      });
      userId = newUser.id;
    } else {
      // Validar que el User existe (solo si se proporciona)
      const user = await prisma.user.findUnique({
        where: { id: userId },
      });
      if (!user) {
        throw new Error('User not found');
      }
    }

    // Crear ClientProfile con el userId
    const profile = await prisma.clientProfile.create({
      data: {
        userId,
        // ... resto de campos
      },
    });




    return { success: true, data: profile };
  } catch (error) {
    // ...
  }
}
```

2. Modificar Frontend





```
// NO enviar userId en el body
const response = await fetch('/api/clients/profiles', {
  method: 'POST',
  body: JSON.stringify({
    // userId: session.user.id, ✗ ELIMINAR ESTA LÍNEA
    firstName: data.firstName,
    lastName: data.lastName,
    // ... resto de campos
  }),
});
```

IMPACTO Y ALCANCE



Funcionalidades Bloqueadas Actualmente:

-  Crear nuevos clientes desde la interfaz
-  Agregar historial de clientes
-  Crear notas de clientes (parcialmente)

Funcionalidades NO Afectadas:

-  Listar clientes existentes (si los hay)
-  Editar clientes existentes
-  Eliminar clientes
-  Productos, Inventario, Ventas (no dependen de ClientProfile)

Riesgo de Regresión:

-  MEDIO: Los cambios afectan solo al módulo de clientes
-  BAJO: Otros módulos no tienen este patrón de validación

ESTADÍSTICAS DEL ANÁLISIS

- **Archivos analizados:** 25+
- **Módulos revisados:** 7 (Clientes, Productos, Inventario, Ventas, Servicios, Notificaciones, Reportes)
- **Archivos con error "User not found":** 4
 - `clientManager.ts` (1 ocurrencia)
 - `historyService.ts` (2 ocurrencias)
 - `noteManager.ts` (1 ocurrencia)
- **Endpoints afectados:** 3
 - `POST /api/clients/profiles`
 - `GET /api/clients/profiles/[id]/history`
 - `POST /api/clients/notes`

PRÓXIMOS PASOS RECOMENDADOS

Fase 1: Decisión de Arquitectura (URGENTE)

1. **Decidir qué modelo usar:**
 - ¿Usar `Client` (simple, recomendado)?
 - ¿Usar `ClientProfile` + crear Users automáticamente?
 - ¿Migrar todo a un modelo unificado?

Fase 2: Implementación del Fix

1. Aplicar la solución elegida
2. Actualizar frontend y backend
3. Probar flujo completo de creación de clientes

Fase 3: Validación

1. Verificar que se pueden crear clientes
2. Verificar que se puede agregar historial
3. Verificar que se pueden crear notas
4. Probar integración con Citas y Ventas

Fase 4: Documentación

1. Documentar la arquitectura elegida
2. Actualizar guías de desarrollo
3. Crear tests para prevenir regresiones



NOTAS ADICIONALES

Observaciones Importantes:

1. **Dualidad de Modelos:** La existencia de `Client` y `ClientProfile` sugiere una evolución del sistema donde:
 - `Client` era el modelo original (Fase 1)
 - `ClientProfile` se agregó en Fase 2 para CRM avanzado
 - Pero no se completó la integración entre ambos
2. **Rol CLIENT en User:** El enum `UserRole` incluye un rol `CLIENT`, lo que sugiere que originalmente se planeó que los clientes tuvieran cuentas de usuario, pero esto nunca se implementó completamente.
3. **Inconsistencia en la UI:** La interfaz muestra “Nuevo Cliente” pero internamente intenta crear un `ClientProfile`, no un `Client`.
4. **Migración de Datos:** Si se decide usar `Client` en lugar de `ClientProfile`, se debe considerar:
 - ¿Hay datos existentes en `ClientProfile`?
 - ¿Se necesita migración?
 - ¿Se mantienen ambos modelos?



REFERENCIAS

- **Repositorio:** <https://github.com/qhosting/citaplanner>
- **Commit actual:** ba4c75b
- **PRs relacionados:**
 - PR #76: Implementación completa del frontend de clientes
 - PR #77: Traducción completa al español
 - PR #78: Fix de inconsistencia en respuestas de API
 - PR #79: Estandarización de endpoints
 - PR #80: Corrección de enum Gender

Generado por: Análisis automatizado de código

Fecha: 8 de octubre de 2025

Versión del reporte: 1.0