Ejemplos Prácticos de Uso de Chatwoot en CitaPlanner

📚 Tabla de Contenidos

---

1. Envío de Notificaciones Básicas

Usando NotificationManager (Sistema Antiguo - Compatible)

```
import { notificationManager } from '@/lib/notifications/notificationManager';
import { NotificationType, NotificationChannel } from '@prisma/client';

// Enviar mensaje simple por Chatwoot
const result = await notificationManager.sendNotification({
  type: NotificationType.APPOINTMENT_REMINDER,
  channel: NotificationChannel.CHATWOOT, // Canal explícito
  recipient: '+523331234567', // Número de teléfono en formato internacional
  message: 'Hola, tu cita es mañana a las 10:00 AM',
  tenantId: 'tenant-123',
  recipientName: 'Juan Pérez',
});

console.log(result);
// { success: true, logId: 'log-id-123', messageId: 'chatwoot-456' }
```

Usando ChatwootService Directamente

```
import { chatwootService } from '@/lib/notifications/chatwootService';

// Enviar mensaje directamente
const result = await chatwootService.sendChatwoot({
  to: '+523331234567',
  message: 'Hola, ¿cómo estás?',
  tenantId: 'tenant-123',
  clientName: 'Juan Pérez',
});

console.log(result);
// { success: true, messageId: 'chatwoot-msg-789' }
```

Usando NotificationService (Sistema Nuevo)

```
import { notificationService } from '@/lib/services/notificationService';
import { NotificationType, NotificationChannel } from '@prisma/client';

// Enviar notificación con sistema nuevo
const result = await notificationService.sendNotification({
  type: NotificationType.APPOINTMENT_CONFIRMATION,
  channel: NotificationChannel.CHATWOOT,
  recipientId: 'client-456',
  message: 'Tu cita ha sido confirmada',
});

console.log(result);
// { success: true, logId: 'log-xyz', messageId: 'msg-abc' }
```

---

2. Notificaciones de Citas

Confirmación de Cita

```
import { chatwootService } from '@/lib/notifications/chatwootService';

// Enviar confirmación de cita con formato profesional
const result = await chatwootService.sendAppointmentConfirmation({
  to: '+523331234567',
  tenantId: 'tenant-123',
  clientName: 'María González',
  appointmentDate: '15 de Noviembre, 2025',
  appointmentTime: '10:00 AM',
  serviceName: 'Corte de Cabello Premium',
  professionalName: 'Carlos García',
  branchName: 'Sucursal Centro',
});

console.log(result);
// Mensaje enviado:
// ✅ *Cita Confirmada*
//
// Hola María González, tu cita ha sido confirmada.
//
// 📅 Fecha: 15 de Noviembre, 2025
// ⏰ Hora: 10:00 AM
// 💼 Servicio: Corte de Cabello Premium
// 👤 Profesional: Carlos García
// 📍 Sucursal: Sucursal Centro
//
// ¡Te esperamos!
```

Cancelación de Cita

```
// Enviar notificación de cancelación
const result = await chatwootService.sendAppointmentCancellation({
  to: '+523331234567',
  tenantId: 'tenant-123',
  clientName: 'Pedro Martínez',
  appointmentDate: '20 de Noviembre, 2025',
  appointmentTime: '3:00 PM',
  reason: 'El profesional no está disponible',
});
```

Usando NotificationManager con Appointment

```
import { notificationManager } from '@/lib/notifications/notificationManager';
import { NotificationType, NotificationChannel } from '@prisma/client';

// Enviar notificación de cita usando template
const result = await notificationManager.sendAppointmentNotificationByChannel(
  'appointment-id-123', // ID de la cita
  NotificationType.APPOINTMENT_CONFIRMATION,
  NotificationChannel.CHATWOOT,
  'appointment_confirmation_template' // Nombre del template
);
```

---

3. Recordatorios Automáticos

Recordatorio Simple

```
import { chatwootService } from '@/lib/notifications/chatwootService';

// Enviar recordatorio de cita
const result = await chatwootService.sendAppointmentReminder({
  to: '+523331234567',
  tenantId: 'tenant-123',
  clientName: 'Ana López',
  appointmentDate: 'mañana',
  appointmentTime: '2:00 PM',
  serviceName: 'Manicure y Pedicure',
});

// Mensaje enviado:
// ⏰ *Recordatorio de Cita*
//
// Hola Ana López, te recordamos que tienes una cita:
//
// 📅 mañana
// ⏰ 2:00 PM
// 💼 Manicure y Pedicure
//
// ¡No olvides asistir!
```

Recordatorio Programado con Cron Job

```typescript
// app/api/cron/send-reminders/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/lib/prisma';
import { chatwootService } from '@/lib/notifications/chatwootService';
import { addDays, startOfDay, endOfDay, format } from 'date-fns';
import { es } from 'date-fns/locale';

export async function GET(request: NextRequest) {
  // Verificar token de autorización
  const authHeader = request.headers.get('authorization');
  if (authHeader !== `Bearer ${process.env.CRON_SECRET}`) {
    return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
  }

  try {
    // Obtener citas del día siguiente
    const tomorrow = addDays(new Date(), 1);
    const appointments = await prisma.appointment.findMany({
      where: {
        startTime: {
          gte: startOfDay(tomorrow),
          lte: endOfDay(tomorrow),
        },
        status: 'CONFIRMED',
        reminderSentAt: null, // No se ha enviado recordatorio
      },
      include: {
        client: true,
        service: true,
        professional: true,
        tenant: true,
      },
    });

    const results = [];

    for (const apt of appointments) {
      // Verificar si el tenant tiene Chatwoot habilitado
      const settings = await prisma.notificationSettings.findUnique({
        where: { tenantId: apt.tenantId },
      });

      if (!settings?.chatwootEnabled) {
        continue; // Skip si Chatwoot no está habilitado
      }

      // Enviar recordatorio
      const result = await chatwootService.sendAppointmentReminder({
        to: apt.client.phone,
        tenantId: apt.tenantId,
        clientName: `${apt.client.firstName} ${apt.client.lastName}`,
        appointmentDate: format(apt.startTime, 'dd/MM/yyyy', { locale: es }),
        appointmentTime: format(apt.startTime, 'HH:mm'),
        serviceName: apt.service.name,
      });

      if (result.success) {
        // Marcar como enviado
        await prisma.appointment.update({
          where: { id: apt.id },
          data: { reminderSentAt: new Date() },
        });
```

```
        results.push({ appointmentId: apt.id, success: true });
      } else {
        results.push({ appointmentId: apt.id, success: false, error: result.error });
      }

      // Rate limiting: esperar 100ms entre mensajes
      await new Promise(resolve => setTimeout(resolve, 100));
    }

    return NextResponse.json({
      success: true,
      processed: results.length,
      sent: results.filter(r => r.success).length,
      failed: results.filter(r => !r.success).length,
      details: results,
    });
  } catch (error: any) {
    console.error('Error sending reminders:', error);
    return NextResponse.json(
      { error: error.message },
      { status: 500 }
    );
  }
}
```

---

4. Mensajes de Marketing

Campaña de Descuento

```
import { chatwootService } from '@/lib/notifications/chatwootService';

// Enviar mensaje de campaña a un cliente
const result = await chatwootService.sendMarketingMessage({
  to: '+523331234567',
  tenantId: 'tenant-123',
  clientName: 'Laura Rodríguez',
  campaignMessage: `
🎉 *¡Oferta Especial para Ti!*

Regresa con nosotros y obtén:
✨ 25% de descuento en tu próxima cita
✨ Servicio de cortesía incluido
✨ Válido hasta el 30 de noviembre

Reserva ahora: https://citaplanner.com/book

*Términos y condiciones aplican
  `.trim(),
});
```

Campaña Masiva con Segmentación

```
// app/api/marketing/campaigns/send/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/lib/prisma';
import { chatwootService } from '@/lib/notifications/chatwootService';
import { subMonths } from 'date-fns';
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth-options';

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

  const { campaignMessage, targetSegment, tenantId } = await request.json();

  // Validar entrada
  if (!campaignMessage || !targetSegment || !tenantId) {
    return NextResponse.json(
      { error: 'campaignMessage, targetSegment, and tenantId are required' },
      { status: 400 }
    );
  }

  // Obtener clientes según segmento
  let clients;
  switch (targetSegment) {
    case 'all':
      clients = await prisma.client.findMany({
        where: {
          tenantId,
          phone: { not: null },
          isActive: true,
        },
      });
      break;

    case 'vip':
      clients = await prisma.client.findMany({
        where: {
          tenantId,
          phone: { not: null },
          isActive: true,
          // Clientes con más de 10 citas
          appointments: {
            some: {},
          },
        },
        include: {
          _count: {
            select: { appointments: true },
          },
        },
      });
      clients = clients.filter(c => c._count.appointments >= 10);
      break;

    case 'inactive':
      // Clientes sin citas en últimos 3 meses
      const threeMonthsAgo = subMonths(new Date(), 3);
      clients = await prisma.client.findMany({
        where: {
```

```
        tenantId,
        phone: { not: null },
        isActive: true,
        OR: [
          {
            appointments: {
              none: {
                startTime: { gte: threeMonthsAgo },
              },
            },
          },
          {
            appointments: {
              none: {},
            },
          },
        ],
      },
    });
    break;

  default:
    return NextResponse.json(
      { error: 'Invalid target segment' },
      { status: 400 }
    );
}

const results = [];

// Enviar mensajes con rate limiting
for (const client of clients) {
  try {
    const result = await chatwootService.sendMarketingMessage({
      to: client.phone,
      tenantId,
      clientName: `${client.firstName} ${client.lastName}`,
      campaignMessage,
    });

    results.push({
      clientId: client.id,
      clientName: `${client.firstName} ${client.lastName}`,
      success: result.success,
      error: result.error,
    });

    // Rate limiting: esperar 100ms entre mensajes
    await new Promise(resolve => setTimeout(resolve, 100));
  } catch (error: any) {
    results.push({
      clientId: client.id,
      success: false,
      error: error.message,
    });
  }
}

// Registrar campaña
await prisma.marketingCampaign.create({
  data: {
    tenantId,
    message: campaignMessage,
```

```
      targetSegment,
      sentCount: results.filter(r => r.success).length,
      failedCount: results.filter(r => !r.success).length,
      sentAt: new Date(),
      sentBy: session.user.id,
      channel: 'CHATWOOT',
    },
  });

  return NextResponse.json({
    success: true,
    totalClients: clients.length,
    sentCount: results.filter(r => r.success).length,
    failedCount: results.filter(r => !r.success).length,
    results,
  });
}
```

---

5. Detección Automática de Clientes

Cómo Funciona

Cuando un cliente envía un mensaje por Chatwoot, el webhook detecta automáticamente al cliente por su número de teléfono y lo vincula.

Configuración del Webhook

1. En Chatwoot, ir a **Settings → Integrations → Webhooks**
2. Agregar nuevo webhook:
   - URL: `https://citaplanner.com/api/webhooks/chatwoot`
   - Events: `message_created`, `conversation_created`, `conversation_updated`

Probar Detección Manual

```
import { chatwootClientMatcher } from '@/lib/chatwoot/client-matcher';

// Detectar cliente desde contacto de Chatwoot
const result = await chatwootClientMatcher.matchClientFromChatwootContact(
  12345, // ID del contacto en Chatwoot
  '+523331234567', // Número de teléfono
  'tenant-123', // ID del tenant
  'Juan Pérez' // Nombre del contacto (opcional)
);

console.log(result);
// {
//   matched: true,
//   client: { id: 'client-abc', firstName: 'Juan', ... },
//   action: 'found',
//   message: 'Cliente existente vinculado: Juan Pérez'
// }
```

Sincronización Masiva de Clientes

```
import { chatwootClientMatcher } from '@/lib/chatwoot/client-matcher';

// Sincronizar todos los clientes de un tenant con Chatwoot
const result = await chatwootClientMatcher.syncAllClientsToChat woot('tenant-123');

console.log(result);
// {
//   total: 150,
//   synced: 145,
//   errors: 5
// }
```

---

6. Testing y Validación

Test de Conexión

```
import { chatwootService } from '@/lib/notifications/chatwootService';

// Probar conexión para un tenant
const isConnected = await chatwootService.testConnection('tenant-123');

if (isConnected) {
  console.log('✅ Chatwoot está configurado y funcionando');
} else {
  console.log('❌ Chatwoot no está configurado o hay un problema');
}
```

Test de Todos los Canales

```
import { notificationManager } from '@/lib/notifications/notificationManager';

// Probar todos los canales de notificación
const status = await notificationManager.testAllChannels('tenant-123');

console.log(status);
// {
//   email: true,
//   sms: true,
//   whatsapp: true,
//   chatwoot: true
// }
```

Test de Envío Completo

```
// Test de envío de mensaje completo
async function testChatwootIntegration() {
  console.log('🧪 Iniciando test de integración de Chatwoot...');

  const testTenantId = 'test-tenant-123';
  const testPhone = '+523331234567';

  // 1. Test de conexión
  console.log('\n1️⃣ Test de conexión...');
  const isConnected = await chatwootService.testConnection(testTenantId);
  console.log(`   Resultado: ${isConnected ? '✅' : '❌'}`);

  if (!isConnected) {
    console.log('   ❌ Chatwoot no está configurado. Abortando tests.');
    return;
  }

  // 2. Test de envío simple
  console.log('\n2️⃣ Test de envío simple...');
  const simpleResult = await chatwootService.sendChatwoot({
    to: testPhone,
    message: '🧪 Mensaje de prueba desde CitaPlanner',
    tenantId: testTenantId,
    clientName: 'Usuario de Prueba',
  });
  console.log(`   Resultado: ${simpleResult.success ? '✅' : '❌'}`);
  if (!simpleResult.success) {
    console.log(`   Error: ${simpleResult.error}`);
  }

  // 3. Test de confirmación de cita
  console.log('\n3️⃣ Test de confirmación de cita...');
  const confirmResult = await chatwootService.sendAppointmentConfirmation({
    to: testPhone,
    tenantId: testTenantId,
    clientName: 'Usuario de Prueba',
    appointmentDate: '15 de Noviembre, 2025',
    appointmentTime: '10:00 AM',
    serviceName: 'Servicio de Prueba',
    professionalName: 'Profesional de Prueba',
    branchName: 'Sucursal de Prueba',
  });
  console.log(`   Resultado: ${confirmResult.success ? '✅' : '❌'}`);

  // 4. Test de recordatorio
  console.log('\n4️⃣ Test de recordatorio...');
  const reminderResult = await chatwootService.sendAppointmentReminder({
    to: testPhone,
    tenantId: testTenantId,
    clientName: 'Usuario de Prueba',
    appointmentDate: 'mañana',
    appointmentTime: '2:00 PM',
    serviceName: 'Servicio de Prueba',
  });
  console.log(`   Resultado: ${reminderResult.success ? '✅' : '❌'}`);

  console.log('\n✅ Tests completados');
}

// Ejecutar tests
testChatwootIntegration();
```

7. Casos de Uso Avanzados

Solicitud de Feedback Post-Cita

```
import { chatwootService } from '@/lib/notifications/chatwootService';

// Enviar solicitud de feedback 2 horas después de la cita
const result = await chatwootService.sendFeedbackRequest({
  to: '+523331234567',
  tenantId: 'tenant-123',
  clientName: 'Roberto Sánchez',
  serviceName: 'Corte de Cabello',
  feedbackUrl: 'https://citaplanner.com/feedback/apt-789',
});
```

Notificación con Detección Automática

```typescript
import { prisma } from '@/lib/prisma';
import { chatwootApiService } from '@/lib/chatwoot/api';

// Enviar mensaje y crear/actualizar contacto automáticamente
async function sendWithAutoDetection(
  phoneNumber: string,
  message: string,
  tenantId: string
) {
  // 1. Buscar cliente en BD
  const client = await prisma.client.findFirst({
    where: {
      phone: phoneNumber,
      tenantId,
    },
  });

  if (!client) {
    console.log('Cliente no encontrado en BD');
    return { success: false, error: 'Cliente no encontrado' };
  }

  // 2. Configurar servicio
  await chatwootApiService.loadConfigForTenant(tenantId);

  // 3. Buscar o crear contacto en Chatwoot
  const contact = await chatwootApiService.findOrCreateContact({
    name: `${client.firstName} ${client.lastName}`,
    phone_number: phoneNumber,
    email: client.email || undefined,
    identifier: client.id,
    custom_attributes: {
      client_id: client.id,
      tenant_id: tenantId,
    },
  });

  if (!contact) {
    return { success: false, error: 'No se pudo crear contacto en Chatwoot' };
  }

  // 4. Actualizar cliente con chatwootContactId si no lo tiene
  if (!client.chatwootContactId) {
    await prisma.client.update({
      where: { id: client.id },
      data: { chatwootContactId: contact.id.toString() },
    });
  }

  // 5. Enviar mensaje
  const result = await chatwootApiService.sendMessageToContact({
    to: phoneNumber,
    message,
    tenantId,
  });

  return result;
}

// Uso
await sendWithAutoDetection(
  '+523331234567',
```

```
  'Tu cita está confirmada',
  'tenant-123'
);
```

Mensaje con Botones Interactivos (Futuro)

```
// NOTA: Esta funcionalidad requiere configuración adicional en Chatwoot
// y solo funciona con canales que soporten interactividad (WhatsApp Business API)

import { chatwootApiService } from '@/lib/chatwoot/api';

// Enviar mensaje con botones
const messagePayload = {
  content: 'Confirma tu asistencia a la cita',
  content_type: 'input_select',
  content_attributes: {
    items: [
      { title: 'Confirmar ✅', value: 'confirm' },
      { title: 'Reagendar 📅', value: 'reschedule' },
      { title: 'Cancelar ❌', value: 'cancel' },
    ],
  },
  private: false,
};

// Este sería el flujo dentro del API service
// (pendiente de implementación completa)
```

📌 Notas Importantes

Rate Limiting

- Chatwoot tiene límites de peticiones (60/minuto por defecto)
- Usar delays entre mensajes masivos: `await new Promise(resolve => setTimeout(resolve, 100))`

Formato de Teléfonos

- Siempre usar formato internacional: `+[código país][número]`
- Ejemplo México: `+523331234567`
- El servicio normaliza automáticamente

Seguridad

- Tokens API en variables de entorno, NUNCA en código
- Validar permisos antes de enviar mensajes masivos
- Implementar rate limiting en endpoints públicos

Multi-tenant

- Cada tenant debe tener su propia configuración de Chatwoot
- Configurar en `NotificationSettings` o `ChatwootConfig`
- La configuración puede ser diferente por sucursal

Logging

- Todas las notificaciones se registran en `NotificationLog`

- Incluye: timestamp, estado, error (si aplica), messageId
- Útil para auditoría y debugging

🆘 Troubleshooting

Mensaje no se envía

1. Verificar configuración:

```
const isConfigured = await chatwootService.testConnection('tenant-id');
```

1. Revisar variables de entorno
2. Verificar que el inbox ID sea correcto
3. Revisar logs: `NotificationLog` en BD

Cliente no se detecta automáticamente

1. Verificar que el webhook esté configurado en Chatwoot
2. Verificar que el número de teléfono sea correcto
3. Revisar normalización de números en BD

Contacto no se crea en Chatwoot

1. Verificar que el inbox ID sea correcto
2. Verificar permisos del API token
3. Verificar que el formato del teléfono sea E.164

🎓 Recursos Adicionales

- **Documentación Completa**: `CHATWOOT_NOTIFICATIONS_INTEGRATION.md`
- **Investigación**: `CHATWOOT_INTEGRATION_RESEARCH.md`
- **Resumen**: `CHATWOOT_INTEGRATION_SUMMARY.md`
- **PR en GitHub**: https://github.com/qhosting/citaplanner/pull/119

¡Listo para usar Chatwoot en CitaPlanner! 🚀