

# Phase 1: System Configuration - CitaPlanner

---

## Overview

---

Phase 1 adds comprehensive system configuration capabilities to CitaPlanner, including service categories, notification templates, and multi-channel notification support (Email, SMS, WhatsApp).

## Features Implemented

---

### 1. Service Categories

- Organize services into categories for better management
- Color-coded categories for visual organization
- Optional categorization (services can exist without categories)

### 2. Notification System

- **Multi-channel support:** Email, SMS, WhatsApp
- **Template-based notifications:** Create reusable notification templates
- **Notification logging:** Track all sent notifications with status
- **Variable substitution:** Dynamic content in templates

### 3. Database Schema

#### New Tables

- `service_categories` : Service categorization
- `notification_templates` : Reusable notification templates
- `notification_logs` : Audit trail of all notifications

#### Modified Tables

- `services` : Added optional `categoryId` field
- `appointments` : Added `notificationsSent` boolean field

## API Endpoints

---

### Services

- `GET /api/services` - List all services
- `POST /api/services` - Create new service
- `GET /api/services/[id]` - Get service details
- `PUT /api/services/[id]` - Update service
- `DELETE /api/services/[id]` - Delete service

### Service Categories

- `GET /api/services/categories` - List all categories
- `POST /api/services/categories` - Create new category
- `GET /api/services/categories/[id]` - Get category details
- `PUT /api/services/categories/[id]` - Update category

- DELETE /api/services/categories/[id] - Delete category

## Notifications

- GET /api/notifications/templates - List templates
- POST /api/notifications/templates - Create template
- GET /api/notifications/templates/[id] - Get template
- PUT /api/notifications/templates/[id] - Update template
- DELETE /api/notifications/templates/[id] - Delete template
- POST /api/notifications/send - Send notification manually
- GET /api/notifications/logs - View notification history
- POST /api/notifications/test - Test notification channels

## Environment Variables

Add these to your `.env` file:

```
# Email Configuration (SMTP)
SMTP_HOST="smtp.gmail.com"
SMTP_PORT="587"
SMTP_USER="your-email@gmail.com"
SMTP_PASSWORD="your-app-password"
SMTP_FROM="noreply@citaplanner.com"

# SMS Configuration (Twilio - Optional)
TWILIO_ACCOUNT_SID="your-twilio-account-sid"
TWILIO_AUTH_TOKEN="your-twilio-auth-token"
TWILIO_PHONE_NUMBER="+1234567890"

# WhatsApp Configuration (Evolution API)
EVOLUTION_API_URL="https://your-evolution-api-url.com"
EVOLUTION_API_KEY="your-evolution-api-key"
EVOLUTION_INSTANCE="your-instance-name"
```

## Evolution API Setup

### What is Evolution API?

Evolution API is a RESTful API for WhatsApp messaging that allows you to send and receive WhatsApp messages programmatically.

### Configuration Steps

#### 1. Get Evolution API Instance

- Sign up for Evolution API service
- Create a new instance
- Note your instance name

#### 2. Get API Credentials

- Obtain your API key from the dashboard
- Note your API URL (e.g., `https://api.evolution.com` )

#### 3. Connect WhatsApp

- Use the Evolution API dashboard to connect your WhatsApp number

- Scan the QR code with your WhatsApp mobile app
- Wait for connection confirmation

#### 4. Configure CitaPlanner

- Add credentials to `.env` file:

```
env
EVOLUTION_API_URL="https://your-api-url.com"
EVOLUTION_API_KEY="your-api-key"
EVOLUTION_INSTANCE="your-instance-name"
```

#### 5. Test Connection

- Use the test endpoint: `POST /api/notifications/test`
- Verify WhatsApp connection status

### Evolution API Endpoints Used

- `POST /message/sendText/{instance}` - Send text message

```
json
{
  "number": "5215551234567",
  "text": "Your message here",
  "delay": 0,
  "linkPreview": true
}
```

- `GET /instance/connectionState/{instance}` - Check connection status

### Phone Number Format

- Include country code without + or spaces
- Example: Mexico `5215551234567` (52 + 10-digit number)

## Template Variables

---

Available variables for notification templates:

- `{{clientName}}` - Client's full name
- `{{serviceName}}` - Service name
- `{{appointmentDate}}` - Appointment date
- `{{appointmentTime}}` - Appointment time
- `{{professionalName}}` - Professional's name
- `{{branchName}}` - Branch name
- `{{price}}` - Service price

### Example Template

**Email Template:**


Subject: Confirmación de Cita - {{serviceName}}

Hola {{clientName}},

Tu cita para {{serviceName}} ha sido confirmada.

Detalles:


- Fecha: {{appointmentDate}}
- Hora: {{appointmentTime}}
- Profesional: {{professionalName}}
- Sucursal: {{branchName}}
- Precio: {{price}}

 Te esperamos!

### WhatsApp Template:

Hola {{clientName}}! 🙌

Tu cita está confirmada:

 {{appointmentDate}} a las {{appointmentTime}}

 {{serviceName}}

 Con {{professionalName}}

 {{branchName}}

 {{price}}

¡Nos vemos pronto!

## Usage Examples

### Creating a Service Category

```
const response = await fetch('/api/services/categories', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    name: 'Cortes de Cabello',
    description: 'Servicios de corte y peinado',
    color: '#FF6B6B',
  }),
});
```

### Creating a Service with Category

```
const response = await fetch('/api/services', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    name: 'Corte Caballero',
    description: 'Corte de cabello para caballero',
    price: 250,
    duration: 30,
    categoryId: 'category-id-here',
  }),
});
```

## Sending a Notification

```
const response = await fetch('/api/notifications/send', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    type: 'WHATSAPP',
    recipient: '5215551234567',
    message: 'Tu cita ha sido confirmada para mañana a las 10:00 AM',
    appointmentId: 'appointment-id-here',
  }),
});
```

## Testing Notification Channels

```
const response = await fetch('/api/notifications/test', {
  method: 'POST',
});

const result = await response.json();
// { email: true, sms: false, whatsapp: true }
```

## Non-Breaking Changes

---

All Phase 1 changes are **completely non-breaking**:

1. ☒ New tables only (no modifications to existing tables except optional fields)
2. ☒ Optional `categoryId` in services (nullable)
3. ☒ Optional `notificationsSent` in appointments (default: false)
4. ☒ Existing appointments continue working without modifications
5. ☒ Services work with or without categories
6. ☒ Notifications are opt-in (not automatic)

## Migration

---

The database migration was created and applied:

- Migration name: `20251007200241_phase1_system_configuration`
- All new tables created successfully
- Existing data preserved

## Testing

---

### Build Test

```
cd app
npm run build
```

## Database Test

```
npx prisma studio  
# Verify new tables exist
```

## API Test

```
# Test services endpoint  
curl http://localhost:3000/api/services  
  
# Test categories endpoint  
curl http://localhost:3000/api/services/categories  
  
# Test notification test endpoint  
curl -X POST http://localhost:3000/api/notifications/test
```

## Next Steps

---

After merging Phase 1:

1. **Configure notification channels** in production
2. **Create notification templates** for common scenarios
3. **Test all three channels** (Email, SMS, WhatsApp)
4. **Create service categories** for better organization
5. **Update UI components** to use new features

## Support

---

For issues or questions:

- Check notification logs: `GET /api/notifications/logs`
- Test channels: `POST /api/notifications/test`
- Review Evolution API documentation
- Check environment variables configuration

## Security Notes

---


- API keys should never be committed to repository
- Use environment variables for all credentials
- Evolution API key should be kept secure
- SMTP passwords should use app-specific passwords
- Twilio credentials should be production-grade

---

**Phase 1 Status:**  Complete and Ready for Production

**Breaking Changes:**  None

**Database Migration:**  Applied Successfully

**Build Status:**  Passing