

# Guía de Despliegue - Integraciones Externas

---

## Índice

---

1. [Checklist Pre-Despliegue](#)
  2. [Configuración de Variables de Entorno](#)
  3. [Configuración de Webhooks](#)
  4. [Testing de Integraciones](#)
  5. [Despliegue por Plataforma](#)
  6. [Monitoreo en Producción](#)
  7. [Rollback y Recuperación](#)
  8. [Troubleshooting Común](#)
- 

## Checklist Pre-Despliegue

---

### ✓ Credenciales y Configuración

- ☐ WhatsApp Business API
- ☐ Access Token obtenido
- ☐ Phone Number ID configurado
- ☐ Business Account ID configurado
- ☐ Número de prueba verificado
- ☐ CRM (al menos uno configurado)
- ☐ Salesforce: Access Token + Instance URL
- ☐ HubSpot: Access Token (Private App)
- ☐ Zoho: Access Token + API Domain
- ☐ Campos personalizados creados
- ☐ Mercado Pago
- ☐ Credenciales de TEST obtenidas
- ☐ Webhooks configurados
- ☐ Testing en sandbox completado
- ☐ Credenciales de PRODUCCIÓN obtenidas
- ☐ Cuenta certificada (para producción)

### ✓ Base de Datos

- ☐ Tablas de Supabase creadas:
- ☐ `payment_preferences`
- ☐ `transactions`
- ☐ `wallet_transactions`
- ☐ `payment_history`

- [ ] `user_notification_settings`

```
-- Crear tablas necesarias para integraciones
CREATE TABLE IF NOT EXISTS payment_preferences (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  preference_id TEXT UNIQUE NOT NULL,
  debt_id UUID REFERENCES debts(id),
  debtor_id UUID REFERENCES users(id),
  amount DECIMAL(12, 2) NOT NULL,
  external_reference TEXT,
  status TEXT DEFAULT 'pending',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE TABLE IF NOT EXISTS transactions (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  payment_id TEXT UNIQUE NOT NULL,
  external_reference TEXT,
  status TEXT NOT NULL,
  status_detail TEXT,
  amount DECIMAL(12, 2) NOT NULL,
  currency TEXT DEFAULT 'CLP',
  payment_method TEXT,
  payment_type TEXT,
  payer_email TEXT,
  metadata JSONB,
  date_created TIMESTAMP WITH TIME ZONE,
  date_approved TIMESTAMP WITH TIME ZONE,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

CREATE TABLE IF NOT EXISTS user_notification_settings (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  user_id UUID REFERENCES users(id) UNIQUE NOT NULL,
  phone_number TEXT,
  whatsapp_settings JSONB DEFAULT '{}',
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

## ✓ Código y Build

- [ ] Código de Parte 2B integrado correctamente
- [ ] `npm install` ejecutado sin errores
- [ ] Build de producción exitoso ( `npm run build` )
- [ ] No hay errores de TypeScript/ESLint
- [ ] Tests unitarios pasando

## ✓ Documentación

- [ ] Documentación completa revisada
  - [ ] README actualizado con nuevas funcionalidades
  - [ ] `.env.example` actualizado
  - [ ] Guías de configuración accesibles
-

# Configuración de Variables de Entorno

## Desarrollo ( .env.local )

```
# Supabase
VITE_SUPABASE_URL=https://tu-proyecto.supabase.co
VITE_SUPABASE_ANON_KEY=tu-anon-key

# WhatsApp (TEST)
VITE_WHATSAPP_ACCESS_TOKEN=tu-test-token
VITE_WHATSAPP_PHONE_NUMBER_ID=tu-phone-id
VITE_WHATSAPP_BUSINESS_ACCOUNT_ID=tu-account-id

# Salesforce (TEST/SANDBOX)
VITE_SALESFORCE_ACCESS_TOKEN=tu-sandbox-token
VITE_SALESFORCE_INSTANCE_URL=https://test.salesforce.com

# HubSpot (TEST)
VITE_HUBSPOT_ACCESS_TOKEN=tu-test-token

# Zoho (TEST)
VITE_ZOHO_ACCESS_TOKEN=tu-test-token
VITE_ZOHO_API_DOMAIN=https://www.zohoapis.com

# Mercado Pago (TEST)
VITE_MERCADOPAGO_ACCESS_TOKEN=TEST-123456-102030-xxx
VITE_MERCADOPAGO_PUBLIC_KEY=TEST-xxxxxxxx-xxxx
VITE_MERCADOPAGO_SANDBOX=true

# Environment
VITE_APP_ENV=development
```

## Producción

### Vercel

1. Ve a tu proyecto en Vercel
2. Settings > Environment Variables
3. Agrega cada variable:

```
VITE_SUPABASE_URL = https://tu-proyecto.supabase.co
VITE_SUPABASE_ANON_KEY = tu-anon-key
VITE_WHATSAPP_ACCESS_TOKEN = tu-production-token
...
```

 **Importante:** Marca las variables sensibles como “Encrypted”

### Netlify

1. Ve a tu proyecto en Netlify
2. Site settings > Environment variables
3. Agrega variables:

```
VITE_SUPABASE_URL = https://tu-proyecto.supabase.co
VITE_SUPABASE_ANON_KEY = tu-anon-key
...
```

## Railway

```
# Usando Railway CLI
railway variables set VITE_SUPABASE_URL=https://tu-proyecto.supabase.co
railway variables set VITE_MERCADOPAGO_ACCESS_TOKEN=APP-xxx
```

## Docker

```
# docker-compose.yml
version: '3.8'
services:
  app:
    build: .
    environment:
      - VITE_SUPABASE_URL=${VITE_SUPABASE_URL}
      - VITE_MERCADOPAGO_ACCESS_TOKEN=${VITE_MERCADOPAGO_ACCESS_TOKEN}
      # ... más variables
    ports:
      - "3000:3000"
```

# Configuración de Webhooks

## Mercado Pago Webhooks

### Desarrollo

1. Usa ngrok para exponer tu servidor local:

```
ngrok http 5173
```

1. Copia la URL generada (ej: `https://abc123.ngrok.io` )
2. En Mercado Pago:
  - Ve a Tus integraciones > Webhooks
  - Agrega URL: `https://abc123.ngrok.io/api/webhooks/mercadopago`
  - Eventos: `payment`

### Producción

1. URL de webhook: `https://tudominio.com/api/webhooks/mercadopago`
2. Crea el endpoint en tu backend:

```
// Express.js example
import express from 'express';
import mercadoPagoService from './services/integrations/mercadopago.service';

const app = express();
app.use(express.json());

app.post('/api/webhooks/mercadopago', async (req, res) => {
  try {
    console.log('📦 Webhook recibido:', req.body);

    // Procesar webhook
    await mercadoPagoService.processWebhook(req.body);

    // Responder inmediatamente
    res.status(200).send('OK');
  } catch (error) {
    console.error('❌ Error procesando webhook:', error);
    res.status(500).send('Error');
  }
});

app.listen(3001, () => {
  console.log('Webhook server running on port 3001');
});
```

#### 1. Configura en Mercado Pago:

- URL: `https://tudominio.com/api/webhooks/mercadopago`
- Eventos: `payment`, `merchant_order`

## Verificar Webhooks

```
# Test webhook localmente
curl -X POST http://localhost:3001/api/webhooks/mercadopago \
  -H "Content-Type: application/json" \
  -d '{
    "action": "payment.created",
    "api_version": "v1",
    "data": {"id": "123456789"},
    "type": "payment"
  }'
```

# Testing de Integraciones

## 1. WhatsApp

```
// test-whatsapp.js
import whatsappService from './services/integrations/whatsapp.service';

async function testWhatsApp() {
  console.log('🔧 Testing WhatsApp...');

  // 1. Verificar configuración
  const config = whatsappService.isConfigured();
  console.log('Configuración:', config);

  if (!config.configured) {
    console.error('❌ WhatsApp no configurado');
    return;
  }

  // 2. Enviar mensaje de prueba
  const result = await whatsappService.sendMessage(
    'TU_NUMERO_DE_PRUEBA',
    '🔧 Mensaje de prueba desde testing'
  );

  console.log('Resultado:', result);

  if (result.success) {
    console.log('✅ WhatsApp funciona correctamente');
  } else {
    console.error('❌ Error:', result.error);
  }
}

testWhatsApp();
```

## 2. CRM

```
// test-crm.js
import crmService from './services/integrations/crm/crm.service';

async function testCRM() {
  console.log('🔧 Testing CRM...');

  // 1. Verificar CRMs disponibles
  const crms = crmService.getAvailableCRMs();
  console.log('CRMs disponibles:', crms);

  // 2. Sincronizar un deudor de prueba
  const result = await crmService.syncDebtor({
    firstName: 'Test',
    lastName: 'User',
    email: 'test@example.com',
    phone: '56912345678',
    rut: '12345678-9',
    totalDebt: 100000
  });

  console.log('Resultado sincronización:', result);

  if (result.success) {
    console.log('✅ CRM funciona correctamente');
  } else {
    console.error('❌ Error:', result.error);
  }
}

testCRM();
```

### 3. Mercado Pago

```
// test-mercadopago.js
import mercadoPagoService from './services/integrations/mercadopago.service';

async function testMercadoPago() {
  console.log('🔧 Testing Mercado Pago...');

  // 1. Verificar configuración
  const config = mercadoPagoService.isConfigured();
  console.log('Configuración:', config);

  // 2. Crear preferencia de prueba
  const result = await mercadoPagoService.createPaymentPreference({
    debtId: 'test-123',
    debtorId: 'user-456',
    debtorEmail: 'test@example.com',
    debtorName: 'Test User',
    amount: 10000,
    description: 'Pago de prueba'
  });

  console.log('Resultado:', result);

  if (result.success) {
    console.log('✅ Mercado Pago funciona');
    console.log('🔗 URL de pago:', result.initPoint);
  } else {
    console.error('❌ Error:', result.error);
  }
}

testMercadoPago();
```

## Despliegue por Plataforma

### Vercel

#### 1. Conectar Repositorio:

```
# Instalar Vercel CLI
npm i -g vercel

# Login
vercel login

# Deploy
vercel
```

#### 1. Configurar Variables:

- Ve a Settings > Environment Variables
- Agrega todas las variables de `.env.example`

#### 2. Configurar Build:



```
// vercel.json
{
  "buildCommand": "npm run build",
  "outputDirectory": "dist",
  "framework": "vite"
}
```

### 1. Deploy a Producción:

```
vercel --prod
```

## Netlify

### 1. Conectar Repositorio:

- Ve a [Netlify](https://app.netlify.com/) (https://app.netlify.com/)
- New site from Git
- Selecciona tu repositorio

### 2. Configurar Build:

- Build command: `npm run build`
- Publish directory: `dist`

### 3. Variables de Entorno:

- Site settings > Environment variables
- Agrega todas las variables

### 4. Deploy:

- Push a la rama principal
- Deploy automático

## Railway

### 1. Crear Proyecto:

```
# Instalar Railway CLI
npm i -g @railway/cli

# Login
railway login

# Inicializar
railway init

# Deploy
railway up
```

### 1. Variables de Entorno:

```
railway variables set VITE_SUPABASE_URL=xxx
railway variables set VITE_MERCADOPAGO_ACCESS_TOKEN=xxx
# ... más variables
```

## AWS Amplify

### 1. Conectar Repositorio:

- Ve a AWS Amplify Console
- New app > Host web app
- Conecta Git

### 2. Configurar Build:

```
# amplify.yml
version: 1
frontend:
  phases:
    preBuild:
      commands:
        - npm ci
    build:
      commands:
        - npm run build
  artifacts:
    baseDirectory: dist
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
```

### 1. Variables de Entorno:

- App settings > Environment variables
- Agrega variables

---

## Monitoreo en Producción

### 1. Logs

#### Vercel




```
# Ver logs en tiempo real
vercel logs --follow
```

#### Netlify

```
# En la UI: Deploys > Function logs
```

### 2. Alertas

Configura alertas para:

-  Errores en webhooks
-  Pagos fallidos
-  Métricas de uso

```
// Ejemplo con Sentry
import * as Sentry from '@sentry/react';

Sentry.init({
  dsn: 'tu-sentry-dsn',
  environment: process.env.VITE_APP_ENV,
  integrations: [
    new Sentry.BrowserTracing(),
    new Sentry.Replay()
  ],
  tracesSampleRate: 1.0
});

// Reportar errores
try {
  await operation();
} catch (error) {
  Sentry.captureException(error);
  throw error;
}
```

### 3. Métricas

```
// Tracking de eventos importantes
function trackEvent(eventName, properties) {
  // Google Analytics
  if (window.gtag) {
    window.gtag('event', eventName, properties);
  }

  // Mixpanel
  if (window.mixpanel) {
    window.mixpanel.track(eventName, properties);
  }
}

// Ejemplos
trackEvent('payment_created', {
  amount: 50000,
  method: 'mercadopago'
});

trackEvent('whatsapp_sent', {
  type: 'payment_reminder'
});

trackEvent('crm_sync_completed', {
  crm: 'salesforce',
  debtors: 150
});
```

## 4. Health Checks

```
// /api/health endpoint
app.get('/api/health', async (req, res) => {
  const health = {
    status: 'healthy',
    timestamp: new Date().toISOString(),
    integrations: {
      whatsapp: whatsappService.isConfigured().configured,
      crm: crmService.getAvailableCRMs().some(c => c.configured),
      mercadopago: mercadoPagoService.isConfigured().configured
    }
  };

  const allHealthy = Object.values(health.integrations).every(v => v);

  res.status(allHealthy ? 200 : 503).json(health);
});
```

---

## Rollback y Recuperación

### Plan de Rollback

#### 1. Rollback Inmediato:

```
# Vercel
vercel rollback

# Netlify
# UI: Deploys > Previous deploy > Publish
```

#### 1. Rollback con Git:

```
# Revertir último commit
git revert HEAD
git push

# O volver a commit específico
git reset --hard <commit-hash>
git push -f
```

## Backup de Datos

```
// Script de backup
async function backupCriticalData() {
  // 1. Exportar preferencias de pago pendientes
  const { data: preferences } = await supabase
    .from('payment_preferences')
    .select('*')
    .eq('status', 'pending');

  // 2. Guardar en archivo
  fs.writeFileSync(
    `backup-${Date.now()}.json`,
    JSON.stringify(preferences, null, 2)
  );

  console.log('✅ Backup completado');
}
```

## Recuperación ante Fallas

```
// Reintento automático con backoff exponencial
async function retryWithBackoff(fn, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      return await fn();
    } catch (error) {
      if (i === maxRetries - 1) throw error;

      const delay = Math.pow(2, i) * 1000;
      console.log(`⌚ Reintentando en ${delay}ms...`);
      await sleep(delay);
    }
  }
}

// Uso
await retryWithBackoff(async () => {
  return await mercadoPagoService.createPaymentPreference(data);
});
```

## Troubleshooting Común

### 1. Variables de Entorno No Reconocidas

**Síntoma:** `undefined` al acceder a `import.meta.env.VITE_*`

**Solución:**

```
# 1. Verificar que empiecen con VITE_
# ❌ MERCADOPAGO_TOKEN
# ✅ VITE_MERCADOPAGO_TOKEN

# 2. Reiniciar servidor de desarrollo
npm run dev

# 3. En producción, reconstruir
npm run build
```

## 2. Webhooks No se Reciben

**Síntoma:** Mercado Pago envía webhook pero no llega a tu servidor

**Checklist:**

```
# 1. Verificar URL es accesible
curl https://tudominio.com/api/webhooks/mercadopago

# 2. Verificar logs del servidor
tail -f /var/log/app.log

# 3. Verificar firewall
sudo ufw allow 443

# 4. Verificar endpoint responde 200
# Debe retornar 200 en < 10 segundos
```

## 3. CRM Sync Falla

**Síntoma:** Sincronización con CRM retorna error

**Debug:**

```
// Activar logs detallados
localStorage.setItem('DEBUG_CRM', 'true');

// Verificar token
const config = crmService.getActiveAdapter().isConfigured();
console.log(config);

// Test de conexión
const test = await crmService.getDebtors({ limit: 1 });
console.log('Test:', test);
```

## 4. Pagos Quedan Pendientes

**Síntoma:** Pagos no se actualizan después de aprobarse

**Investigación:**

```
// 1. Verificar webhook fue recibido
SELECT * FROM transactions
WHERE payment_id = 'xxx';

// 2. Verificar deuda fue actualizada
SELECT * FROM debts
WHERE id = 'debt-id';

// 3. Manualmente procesar el pago
await mercadoPagoService.getPayment('payment-id');
await mercadoPagoService.processApprovedPayment(payment);
```

---

## Checklist Post-Despliegue

- [ ] Todas las integraciones verificadas en producción
- [ ] Webhooks funcionando correctamente
- [ ] Logs y monitoreo activos
- [ ] Alertas configuradas
- [ ] Documentación actualizada
- [ ] Equipo capacitado en nuevas funcionalidades
- [ ] Plan de soporte establecido
- [ ] Métricas baseline capturadas

---

## Soporte y Contacto

- **Documentación:** Ver carpeta `/docs/parte2b/`
- **Issues:** Crear issue en GitHub
- **Email:** soporte@tuempresa.com

---

**Versión:** 1.0.0

**Última actualización:** Octubre 2025