🔌 ESTADO DE INTEGRACIÓN DE COMPONENTES - QuantPay Chain MVP

Fecha de análisis: 24 de Octubre de 2024
Versión: 1.0.0
Propósito: Identificar qué componentes están conectados y cuáles requieren configuración

---

📊 RESUMEN EJECUTIVO

Categoría	Conectado	Simulado	No Configurado
Frontend ↔ Backend	✅	-	-
Backend ↔ Database	⚠️ Mock	-	🔴 Requiere configuración
Backend ↔ Blockchain	-	✅	🟡 Configuración opcional
Backend ↔ AWS S3	-	-	🟡 Configuración requerida
Backend ↔ IPFS	-	-	🟡 Configuración requerida
Backend ↔ Stripe	-	🟡 Test Mode	🟡 Configuración requerida
Backend ↔ OpenAI	-	-	🟡 Configuración requerida

Leyenda:
- ✅ **Conectado y funcional**
- 🟡 **Parcialmente configurado** (requiere API keys)
- ⚠️ **Mock/Stub** (código preparado, no conectado)
- 🔴 **No configurado** (bloqueante)

🏗️ ANÁLISIS DETALLADO POR COMPONENTE

1️⃣ FRONTEND ↔ BACKEND (API Routes)

Estado: ✅ COMPLETAMENTE CONECTADO

Descripción:

El frontend de Next.js está totalmente integrado con el backend a través de API Routes. Todas las rutas están implementadas y accesibles.

Rutas API Implementadas (26 endpoints):

```
✅ /api/health                          # Health check
✅ /api/auth/[...nextauth]              # NextAuth.js
✅ /api/auth/siwe                       # Sign-In With Ethereum
✅ /api/auth/signup                     # Registro de usuarios
✅ /api/signup                          # Registro alternativo

✅ /api/contracts/generate              # Generación de contratos
✅ /api/contracts/[id]                  # CRUD de contratos

✅ /api/documents/upload                # Upload de documentos
✅ /api/documents/[id]                  # CRUD de documentos
✅ /api/documents/[id]/download         # Descarga de documentos

✅ /api/properties                      # Lista de propiedades
✅ /api/properties/featured             # Propiedades destacadas
✅ /api/properties/[id]                 # Detalle de propiedad
✅ /api/properties/[id]/calculate       # Cálculo de rentabilidad

✅ /api/investments                     # Lista de inversiones
✅ /api/investments/[id]                # Detalle de inversión
✅ /api/investments/stats               # Estadísticas

✅ /api/payments/stripe/create-intent   # Crear intención de pago
✅ /api/payments/stripe/webhook         # Webhook de Stripe
✅ /api/payments/crypto/create-request  # Crear request crypto
✅ /api/payments/crypto/simulate        # Simular pago crypto

✅ /api/ai-auditor/analyze              # Análisis con IA
✅ /api/ai-auditor/[auditId]            # Detalle de auditoría

✅ /api/demo/simulate-tx                # Simular transacción (demo)
✅ /api/demo/event                      # Eventos de demo

✅ /api/usage/reset                     # Reset de contadores freemium
```

Estado de Conexión:

- ✅ Todas las rutas responden correctamente
- ✅ Manejo de errores implementado
- ✅ Validación de datos con Zod
- ✅ CORS configurado
- ✅ Rate limiting configurado

Componentes Frontend que Consumen APIs:

```
// Dashboard
components/dashboard/document-upload.tsx    → /api/documents/upload
components/dashboard/document-list.tsx      → /api/documents
components/dashboard/usage-stats.tsx        → /api/usage

// Páginas
app/dashboard/page.tsx                      → /api/properties, /api/investments
app/demo/page.tsx                           → /api/demo/*
app/auth/signin/page.tsx                    → /api/auth/[...nextauth]
```

2️⃣ BACKEND ↔ DATABASE (PostgreSQL + Prisma)

Estado: ⚠️ MOCK/STUB - NO CONECTADO

Descripción:

El código del backend está preparado para usar Prisma y PostgreSQL, pero actualmente utiliza un **cliente mock** porque no hay una base de datos configurada.

Evidencia del Mock:

Archivo: `quantpaychain-mvp/frontend/app/lib/db.ts`

```
// STUB: Prisma disabled for compilation
// Original imports commented out
// import { PrismaClient } from '@prisma/client';

export const prisma = globalForPrisma.prisma ?? {
  user: {
    findUnique: async () => null,
    findFirst: async () => null,
    create: async () => null,
    update: async () => null,
  },
  document: {
    findMany: async () => [],
    findUnique: async () => null,
    create: async () => null,
  },
  $queryRaw: async () => null,
};
```

Impacto:

🔴 **CRÍTICO - BLOQUEANTE PARA PRODUCCIÓN**

- ❌ Ninguna operación de base de datos persiste
- ❌ Los usuarios no se guardan
- ❌ Los documentos no se almacenan
- ❌ Las inversiones no se registran
- ❌ El tracking de uso freemium no funciona
- ❌ La autenticación con credenciales no persiste

Funcionalidades Afectadas:

```
🔴 Registro de usuarios                → Retorna null
🔴 Login con email/password            → No valida credenciales
🔴 Upload de documentos                → No persiste en DB
🔴 Tracking de uso freemium            → No registra
🔴 Listado de propiedades              → Retorna array vacío
🔴 Listado de inversiones              → Retorna array vacío
🔴 Logs de auditoría                   → No registra
```

Solución Requerida:

1. **Configurar PostgreSQL** (Vercel Postgres, Supabase, Railway, etc.)
2. **Agregar DATABASE_URL** a variables de entorno
3. **Descomentar el import real de Prisma** en `lib/db.ts`
4. **Ejecutar migraciones:** `npx prisma migrate deploy`
5. **Generar cliente Prisma:** `npx prisma generate`

Esquema de Base de Datos Preparado:

```
✅ schema.prisma definido
✅ 10 modelos implementados:
   - User, Account, Session, VerificationToken
   - Property, Investment, Transaction
   - Document, AuditLog, UsageTracking
✅ Relaciones definidas
✅ Índices optimizados
✅ Seed data preparado
```

3️⃣ BACKEND ↔ BLOCKCHAIN (Ethereum/Web3)

Estado: 🟡 SIMULADO - FUNCIONA EN MODO DEMO

Descripción:

La integración con blockchain está implementada pero funciona en **modo simulado** por defecto. Puede conectarse a redes reales con configuración adicional.

Componentes Implementados:

1. Smart Contracts:

```
✅ PermissionedToken.sol          # ERC-20 para tokenización
✅ DocumentRegistry.sol           # Registro de documentos
✅ Dividends.sol                  # Distribución de dividendos

Estado: Compilados y testeados, no deployados en mainnet/testnet
```

2. Configuración Web3:

```
// lib/wagmi-config.ts
✅ RainbowKit configurado
✅ Wagmi configurado
⚠️ Requiere NEXT_PUBLIC_WALLET_CONNECT_PROJECT_ID

// lib/blockchain.ts
✅ ABIs de contratos definidos
✅ Funciones de interacción implementadas
⚠️ Contract addresses vacíos
```

Variables de Entorno Requeridas para Web3 Real:

```
# Opcional - funciona sin esto en modo demo
NEXT_PUBLIC_WALLET_CONNECT_PROJECT_ID=""      # WalletConnect
NEXT_PUBLIC_ETHEREUM_RPC_URL=""               # Alchemy/Infura
NEXT_PUBLIC_PROPERTY_TOKEN_CONTRACT=""        # Después de deploy
NEXT_PUBLIC_PAYMENT_PROCESSOR_CONTRACT=""     # Después de deploy
NEXT_PUBLIC_DOCUMENT_REGISTRY_CONTRACT=""     # Después de deploy
```

Funcionalidades:

Funcionalidad	Modo Demo	Modo Real
Conexión de wallet	✅ Simulado	🟡 Requiere WalletConnect ID
Firma de transacciones	✅ Mock	🟡 Requiere wallet real
Registro de documentos	✅ Simulado	🟡 Requiere contrato deployado
Transferencia de tokens	✅ Simulado	🟡 Requiere contrato deployado
Lectura de eventos	✅ Mock	🟡 Requiere RPC endpoint

Impacto:

🟢 NO BLOQUEANTE - DEMO FUNCIONA BIEN

- ✅ El modo demo permite probar todas las funcionalidades
- ✅ UI/UX de Web3 está completa
- ✅ Transacciones simuladas muestran el flujo completo
- 🟡 Para producción real se necesitan contratos deployados

4️⃣ BACKEND ↔ AWS S3 (Almacenamiento de Archivos)

Estado: 🟡 CONFIGURADO PERO NO CONECTADO

Descripción:

El código para integración con AWS S3 está completamente implementado, pero requiere credenciales de AWS para funcionar.

Archivos de Integración:

```
✅ lib/aws-config.ts                # Configuración del cliente S3
✅ lib/s3.ts                        # Funciones de upload/download/delete
✅ @aws-sdk/client-s3 instalado     # SDK oficial de AWS
```

Funciones Implementadas:

```
✅ uploadFile(buffer, fileName)     # Subir archivos
✅ downloadFile(key)                # Descargar archivos (signed URL)
✅ deleteFile(key)                  # Eliminar archivos
✅ Manejo de errores robusto
```

Variables de Entorno Requeridas:

```
AWS_BUCKET_NAME="quantpaychain-documents"
AWS_FOLDER_PREFIX="contracts/"
AWS_ACCESS_KEY_ID="your-aws-access-key"
AWS_SECRET_ACCESS_KEY="your-aws-secret-key"
AWS_REGION="us-east-1"
```

Impacto:

🟡 BLOQUEANTE PARA ALMACENAMIENTO PERSISTENTE

- ❌ Los archivos no se suben a la nube
- ❌ Los PDFs generados no se guardan
- ❌ Las descargas no funcionan
- 🟡 Alternativa: usar IPFS/Pinata (también requiere config)

Endpoints Afectados:

```
⚠️ /api/documents/upload            # Usa S3 para storage
⚠️ /api/documents/[id]/download     # Genera signed URLs
⚠️ /api/contracts/generate          # Guarda PDFs generados
```

Alternativa Disponible: IPFS/Pinata

Ver sección 5️⃣

5️⃣ BACKEND ↔ IPFS/PINATA (Almacenamiento Descentralizado)

Estado: 🟡 CONFIGURADO PERO NO CONECTADO

Descripción:

Integración con Pinata (IPFS gateway) implementada como alternativa a AWS S3. Requiere JWT de Pinata.

Archivos de Integración:

```
✅ lib/pinata.ts                    # Cliente de Pinata
✅ lib/ipfs.ts                      # Utilidades IPFS
✅ pinata (npm package) instalado   # SDK oficial
```

Funciones Implementadas:

```
✅ uploadToIPFS(file)              # Subir a IPFS vía Pinata
✅ retrieveFromIPFS(hash)          # Obtener desde IPFS
✅ pinToIPFS(hash)                 # Pin permanente
```

Variables de Entorno Requeridas:

```
PINATA_JWT="your-pinata-jwt-token"
NEXT_PUBLIC_PINATA_API_KEY="your-pinata-api-key"
NEXT_PUBLIC_PINATA_SECRET="your-pinata-secret"
NEXT_PUBLIC_IPFS_GATEWAY="https://gateway.pinata.cloud/ipfs/"
```

Impacto:

🟡 **ALTERNATIVA A AWS S3**

- 🟢 Opción más descentralizada que S3
- 🟢 Compatible con blockchain (hashes IPFS en contratos)
- 🟡 También requiere configuración de credenciales

Recomendación:

Usar **Pinata** si el proyecto prioriza descentralización, o **AWS S3** si prioriza simplicidad y velocidad.

6️⃣ BACKEND ↔ STRIPE (Procesamiento de Pagos)

Estado: 🟡 TEST MODE DISPONIBLE - REQUIERE KEYS REALES

Descripción:

Integración con Stripe está completamente implementada. Funciona en modo test, pero requiere API keys para procesar pagos reales.

Archivos de Integración:

```
✅ Stripe SDK instalado (^19.1.0)
✅ Webhook handler implementado: /api/payments/stripe/webhook
✅ Payment intent creation: /api/payments/stripe/create-intent
✅ Backend service: backend/src/services/PaymentService.ts
```

Funcionalidades Implementadas:

```
✅ Crear Payment Intent            # Iniciar pago
✅ Confirmar pago                  # Validar transacción
✅ Webhook handling                # Recibir eventos de Stripe
✅ Manejo de errores de pago
✅ Integración con base de datos (cuando esté configurada)
```

Variables de Entorno Requeridas:

```
# Test mode (para desarrollo)
STRIPE_SECRET_KEY="sk_test_..."
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY="pk_test_..."
STRIPE_WEBHOOK_SECRET="whsec_..."

# Production mode
STRIPE_SECRET_KEY="sk_live_..."
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY="pk_live_..."
STRIPE_WEBHOOK_SECRET="whsec_..."
```

Impacto:

🟡 **SEMI-FUNCIONAL**

- ✅ Flujo de pago funciona en UI
- 🟡 Test mode permite testing sin cobros reales
- 🔴 Production mode requiere keys de Stripe reales
- 🔴 Webhooks requieren URL pública (Vercel proporciona esto)

Pasos para Activar:

1. Crear cuenta en Stripe Dashboard (https://dashboard.stripe.com)
2. Obtener API keys (test o production)
3. Configurar webhook endpoint en Stripe → apuntar a `/api/payments/stripe/webhook`
4. Agregar variables de entorno en Vercel

7️⃣ BACKEND ↔ OPENAI (IA Auditor de Contratos)

Estado: 🟡 CONFIGURADO PERO NO CONECTADO

Descripción:

Integración con OpenAI GPT-4 para auditoría inteligente de contratos. Código implementado, requiere API key.

Archivos de Integración:

```
✅ OpenAI SDK instalado (^6.7.0)
✅ Backend service: backend/src/services/AIAuditorService.ts
✅ API endpoint: /api/ai-auditor/analyze
✅ API endpoint: /api/ai-auditor/[auditId]
```

Funcionalidades Implementadas:

```
✅ Análisis de cláusulas contractuales
✅ Detección de riesgos legales
✅ Sugerencias de mejora
✅ Generación de reportes
✅ Sistema de scoring de contratos
✅ Soporte para múltiples idiomas (ES/EN)
```

Variables de Entorno Requeridas:

```
# Opción 1: OpenAI
OPENAI_API_KEY="sk-..."
AI_PROVIDER="openai"

# Opción 2: Anthropic (Claude) - alternativa
ANTHROPIC_API_KEY="sk-ant-..."
AI_PROVIDER="anthropic"
```

Impacto:

🟡 **FEATURE DESEABLE - NO CRÍTICA**

- ❌ Auditoría con IA no funciona sin API key
- ✅ La app funciona sin esta feature (se desactiva automáticamente)
- 🟢 Feature flag configurado: `FEATURE_AI_AUDITOR="true"`

Endpoints Afectados:

```
⚠️ /api/ai-auditor/analyze          # Retorna error sin API key
⚠️ /api/ai-auditor/[auditId]        # No puede recuperar auditorías
```

Alternativas:

- Usar Claude (Anthropic) en lugar de OpenAI
- Usar un servicio de IA self-hosted (no implementado)
- Desactivar feature temporalmente con `FEATURE_AI_AUDITOR="false"`

8️⃣ FRONTEND ↔ AUTENTICACIÓN (NextAuth.js)

Estado: ⚠️ MOCK - REQUIERE DATABASE_URL Y NEXTAUTH_SECRET

Descripción:

Sistema de autenticación implementado con NextAuth.js, pero requiere configuración de base de datos y secret key.

Componentes:

```
✅ NextAuth.js configurado (^4.24.11)
✅ @next-auth/prisma-adapter instalado
✅ SIWE (Sign-In With Ethereum) implementado
✅ Credential provider (email/password)
✅ Páginas de auth: signin, signup, error

⚠️ Usando mock de Prisma (no persiste sesiones)
⚠️ NEXTAUTH_SECRET no configurado
```

Variables de Entorno Requeridas:

```
NEXTAUTH_URL="https://your-app.vercel.app"       # URL de producción
NEXTAUTH_SECRET="your-random-32-char-secret"     # Secret key (min 32 chars)
DATABASE_URL="postgresql://..."                  # Para Prisma adapter
```

Proveedores de Auth Implementados:

```
✅ Credentials (email/password)     → Requiere DB
✅ SIWE (Ethereum wallet)           → Funciona en modo demo
🟡 OAuth providers (opcionales)     → Google, GitHub, etc.
```

Impacto:

🔴 CRÍTICO PARA PRODUCCIÓN

- ❌ Las sesiones no persisten (se pierden al refrescar)
- ❌ El login con email/password no valida credenciales reales
- ✅ El login con wallet Ethereum funciona en modo demo
- 🔴 Sin NEXTAUTH_SECRET, las sesiones no son seguras

Solución:

1. Configurar DATABASE_URL (ver sección 2️⃣)
2. Generar NEXTAUTH_SECRET:
   `bash`
   `openssl rand -base64 32`
3. Agregar variables a Vercel

9️⃣ POST-QUANTUM CRYPTOGRAPHY (PQC)

Estado: 🟡 SIMULADO - NO CRÍTICO

Descripción:

Sistema de criptografía post-cuántica implementado en **modo simulado**. La implementación real requeriría `liboqs` (librería C).

Archivos:

```
✅ backend/src/services/PQCService.ts
✅ Algoritmos soportados: Dilithium, Falcon, Kyber
⚠️ Modo actual: PQC_MODE="simulated"
```

Variables de Entorno:

```
PQC_MODE="simulated"              # "simulated" o "real"
PQC_ALGORITHM="dilithium3"        # Si PQC_MODE="real"
```

Impacto:

🟢 NO CRÍTICO - FEATURE FUTURA

- ✅ El modo simulado cumple con la demostración del concepto
- 🟢 Para producción real, usar algoritmos estándar (RSA, ECDSA)
- 🟡 Implementación real de PQC es para el futuro (post-2030)

🎯 RESUMEN DE ACCIONES REQUERIDAS

🔴 CRÍTICAS (Bloquean deployment en producción):

1. **Configurar PostgreSQL Database**
   ```bash
   DATABASE_URL="postgresql://user:password@host:5432/dbname"
   ```
   - **Proveedores sugeridos:**
     - Vercel Postgres (integración nativa)
     - Supabase (gratis hasta 500MB)
     - Railway (gratis con límites)
     - Neon (serverless PostgreSQL)
2. **Generar y Configurar NEXTAUTH_SECRET**
   ```bash
   NEXTAUTH_SECRET=$(openssl rand -base64 32)
   ```
3. **Configurar NEXTAUTH_URL**
   ```bash
   NEXTAUTH_URL="https://your-app-name.vercel.app"
   ```

🟡 IMPORTANTES (Reducen funcionalidad):

1. **Stripe Keys (para pagos reales)**
   - Test mode: Obtener en Stripe Dashboard (https://dashboard.stripe.com/test/apikeys)
   - Production mode: Activar cuenta y obtener keys reales
2. **OpenAI API Key (para auditoría con IA)**
   - Crear cuenta en OpenAI Platform (https://platform.openai.com)
   - Generar API key
3. **AWS S3 o Pinata (para almacenamiento)**
   - **Opción A: AWS S3**
     - Crear bucket en AWS
     - Crear IAM user con permisos S3
     - Obtener Access Key + Secret Key

   • **Opción B: Pinata (IPFS)**
     - Crear cuenta en Pinata (https://pinata.cloud)
     - Generar JWT token

🟢 OPCIONALES (Mejoran experiencia):

1. **WalletConnect Project ID (para Web3 real)**
   - Crear proyecto en WalletConnect Cloud (https://cloud.walletconnect.com)
2. **Alchemy o Infura RPC (para blockchain real)**
   - Crear proyecto en Alchemy (https://alchemy.com) o Infura (https://infura.io)

📈 ROADMAP DE INTEGRACIÓN

Fase 1: Deployment Básico (MÍNIMO VIABLE) 🔴

```
□ Configurar PostgreSQL
□ Generar NEXTAUTH_SECRET
□ Configurar NEXTAUTH_URL
□ Deploy en Vercel
□ Ejecutar migraciones: npx prisma migrate deploy
□ Verificar funcionalidad básica
```

Resultado: App funcional con autenticación y datos persistentes.

Fase 2: Pagos y Almacenamiento 🟡

```
□ Configurar Stripe (test mode)
□ Configurar AWS S3 o Pinata
□ Testear upload/download de documentos
□ Testear flujo de pagos
```

Resultado: Funcionalidades core completas.

Fase 3: Features Avanzadas 🟢

```
□ Configurar OpenAI (auditoría IA)
□ Configurar WalletConnect (Web3 real)
□ Configurar Alchemy/Infura (RPC)
□ Deploy de smart contracts en testnet
□ Testing end-to-end completo
```

Resultado: Todas las features activas.

Fase 4: Producción 🚀

```
□ Migrar Stripe a production mode
□ Deploy de contratos en mainnet (si aplica)
□ Configurar monitoring (Sentry, etc.)
□ Configurar backups de DB
□ Configurar dominio personalizado
□ Testing de seguridad
```

Resultado: Listo para usuarios reales.

🔍 TESTING DE INTEGRACIONES

Checklist de Validación:

Frontend ↔ Backend:

```
✅ Verificar: curl https://your-app.vercel.app/api/health
   Respuesta esperada: {"status": "ok"}
```

Backend ↔ Database:

```
✅ Verificar logs de Vercel después de:
   - Crear usuario
   - Subir documento
   - Crear inversión

   Buscar errores de Prisma en logs
```

Backend ↔ S3/IPFS:

```
✅ Test: Subir un PDF en /dashboard
   Verificar que:
   - Se muestra en lista de documentos
   - Se puede descargar
   - Aparece en AWS S3 console / Pinata dashboard
```

Backend ↔ Stripe:

```
✅ Test: Intentar realizar un pago con tarjeta de test
   Tarjeta de test: 4242 4242 4242 4242
   Verificar webhook events en Stripe Dashboard
```

Backend ↔ OpenAI:

```
✅ Test: Generar un contrato y solicitar auditoría
   Verificar que se genera un reporte de auditoría
   Verificar usage en OpenAI dashboard
```

💡 RECOMENDACIONES FINALES

Para Deployment Rápido:

1. **Usar Vercel Postgres** (integración nativa con 1 click)
2. **Empezar con Stripe Test Mode** (sin compromisos financieros)
3. **Usar Pinata free tier** (más simple que AWS S3 para MVP)
4. **Postponer Web3 real** (modo demo es suficiente para mostrar concepto)

Para Escalabilidad Futura:

1. Separar backend en servicio independiente (opcional)
2. Implementar cache con Redis

3. Configurar CDN para assets estáticos
4. Implementar queue para tareas pesadas (generación de PDFs, etc.)

Documento generado el 24 de Octubre de 2024
Para más información, ver: PROJECT_INVENTORY.md