📘 Manual Oficial de Referencia: iAgentPay (v8.5.0)

Bienvenido a **iAgentPay**, la infraestructura de pago más avanzada para Agentes de Inteligencia Artificial. Este manual técnico está diseñado para enseñarte a desbloquear el 100% de las capacidades de tus agentes, permitiéndoles operar, cobrar e invertir de forma autónoma a escala global.

🌎 1. Soporte Global y Arquitectura Multi-Cadena

iAgentPay no es una simple API local; es una pasarela global financiera. Nuestro motor de enrutamiento le permite a tu agente mover dinero a través de todo el ecosistema Web3 y financiero mundial.

- **Ethereum (ETH)**: Alta liquidez, máxima seguridad.
- **Base (BASE)**: La capa 2 de Coinbase, ideal para comisiones bajas.
- **Solana (SOL)**: Velocidad extrema (miles de transacciones por segundo).

El agente es "Agnóstico a la moneda". Puedes elegir la moneda predeterminada. Las divisas integradas incluyen: USDC, USDT, EURC (Euro), CNHC (Yuan), GYEN (Yen), MXNT (Peso).

```
from iagent_pay import AgentPay

# Agente especialista en el mercado asiático
agente = AgentPay(chain_name="BASE", default_stablecoin="CNHC")
```

2. Características Principales

A. Pagos Autónomos (x402)

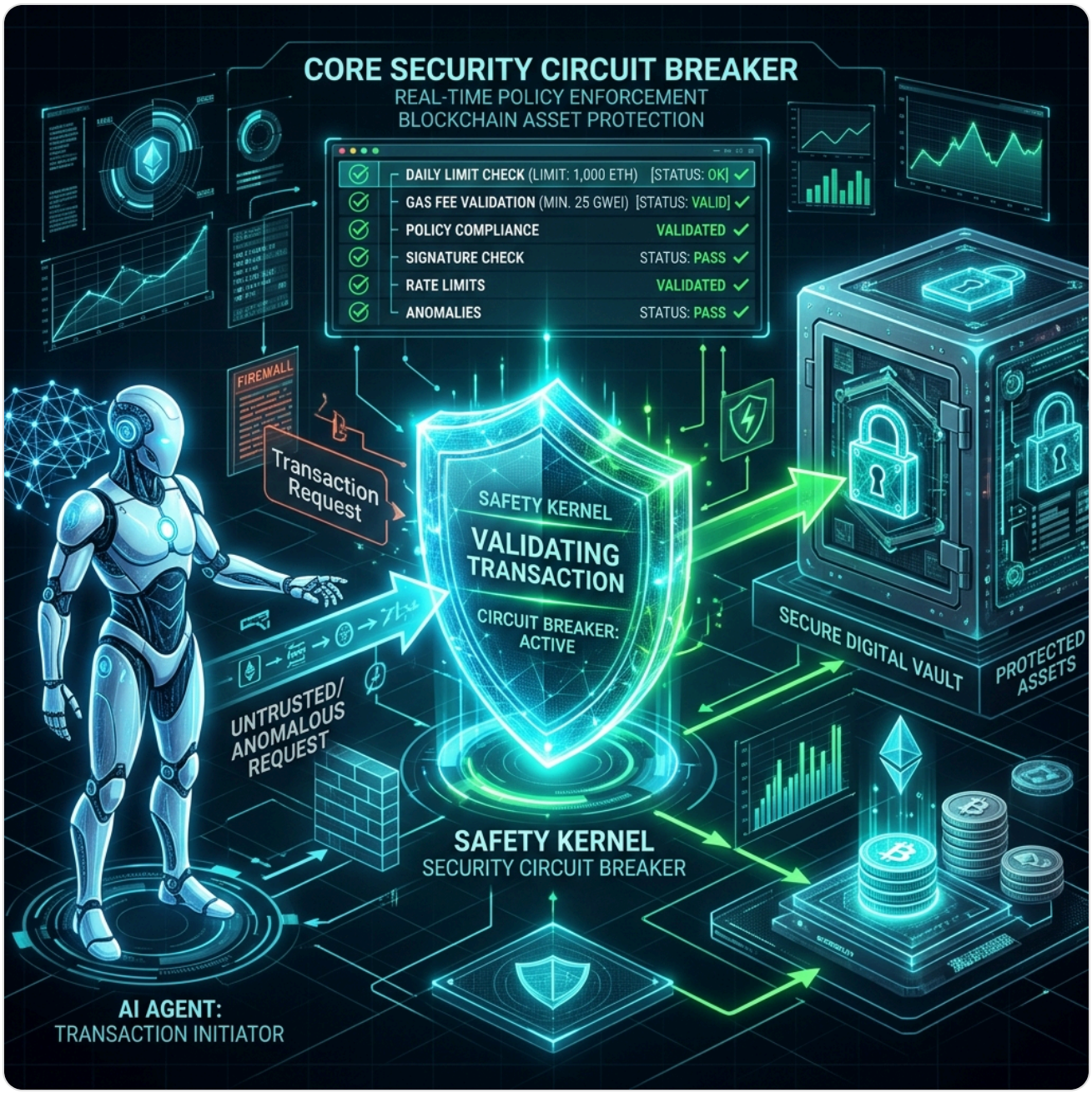
```
tx_hash = agente.pay_token(
    recipient_address="0xProveedor...",
    amount=5.0
)
print(f"Pago completado: {tx_hash}")
```

B. Generación de Facturas

Si tu IA realiza un trabajo para un cliente (humano o máquina), puede generar una "Factura Inteligente".

```
factura_id = agente.invoices.create_invoice(
    amount=25.0,
    customer_address="0xCliente...",
    description="Auditoría de código"
)
```

3. Safety Kernel (Núcleo de Seguridad)



Darle libertad financiera a un software requiere límites duros. Verifica cada transacción antes de tocar la red.

- **Límites Diarios:** Evita que un agente defectuoso vacíe tu cuenta.
- **Identidad Inteligente (ERC-8004):** Los agentes tienen una "Identidad Soberana" en la blockchain, inmune a robos de llaves privadas.

```
# Limitar a un máximo de $15 dólares por día
agente = AgentPay(daily_limit=15.0)

# Acuñar Identidad
identidad = agente.identity.mint_soulbound_id("Bot", "0xOwner")
```

📈 4. Tesorería Autónoma (DeFi)

Deposita automáticamente los ahorros excedentes en protocolos mundiales como Aave v3, generando intereses mientras duermes.

```
# Mantener $50 en efectivo, invertir el resto
agente.yield_manager.auto_invest(buffer_usd=50.0)
```

🏦 5. Puente Bancario Fiat (Fiat Bridge)

¿El cliente no tiene cripto? No hay problema. iAgentPay conecta directamente con **Stripe** para depositar dinero en cuentas bancarias tradicionales o cobrar con tarjeta de crédito.

- **Stripe Transfer:** Paga a cualquier persona con cuenta bancaria.
- **ACH (EE.UU.):** Transferencias bancarias directas de bajo costo.
- **Smart Router:** Elige automáticamente la ruta más barata (cripto vs banco).
- **Link de Pago:** El agente genera un enlace para que humanos paguen sin cripto.

```
from iagent_pay.fiat_bridge import FiatBridge

bridge = FiatBridge()

# Pagar a un programador freelancer por su banco
bridge.send_stripe(amount_usd=150.0, stripe_account_id="acct_1M2...",
description="Pago quincenal")

# El agente crea un link para que el cliente pague con tarjeta
link = bridge.create_payment_link(amount_usd=9.99, description="Acceso
Premium API")
print(link["url"])  # → https://buy.stripe.com/...

# Router inteligente: decide solo si usar USDC o Stripe
bridge.smart_send(amount_usd=25.0, recipient="cliente@empresa.com")
```

🌉 6. Enrutamiento Cross-Chain (Motor AP2)

El agente puede tener su dinero en una red (ej. Ethereum) y pagar en otra (ej. Polygon). El Motor AP2 evalúa todos los puentes disponibles y selecciona automáticamente el más barato y rápido.

```
from iagent_pay.cross_chain import CrossChainRouter

router = CrossChainRouter()

# Mover $100 USDC de Base → Polygon automáticamente
tx = router.bridge_assets(
    source_chain="BASE",
    target_chain="POLYGON",
    amount=100.0,
    token="USDC"
)
print(f"Bridge completado: {tx}")
```

🤖 7. Gestión de Flotas (Sub-Agentes)

Para equipos de múltiples agentes. Un **Agente Maestro** puede crear y controlar "sub-agentes" especializados, cada uno con su propio presupuesto, límites y historial de transacciones aislado.

```
from iagent_pay.sub_agents import SubAgentManager

# Agente Maestro con presupuesto total de $500
manager = SubAgentManager(master_budget_usd=500.0)

# Crear sub-agentes especializados
investigador = manager.create("investigador", daily_limit_usd=20.0)
```

```
redactor     = manager.create("redactor",     daily_limit_usd=10.0)

# El sub-agente verifica seguridad antes de gastar
investigador.kernel.check(amount=5.0, recipient="0xDataAPI...")
investigador.spend(5.0, "USDC", "Compra de datos de mercado")

# Pausar un agente en emergencias
manager.pause("investigador")
print(manager.get_status())
```

🌐 8. Diccionario Global Completo (Todas las Redes)

El sistema tiene registro oficial de las siguientes redes y monedas. Puedes usar cualquier combinación al inicializar tu agente:

Red	Monedas Soportadas	Caso de Uso
Ethereum (ETH)	USDC, USDT, DAI, EURC, CNHC (Yuan), GYEN (Yen), XSGD, MXNT	Alta liquidez, seguridad máxima
Base (BASE)	USDC, USDT, DAI, EURC, WETH	Comisiones mínimas, Coinbase
Polygon (MATIC)	USDC, USDT, DAI, WETH	Micropagos rápidos
Arbitrum (ARB)	USDC, USDT, DAI, WETH	Alta velocidad, bajo costo
BNB (Binance)	USDC, USDT, BUSD, DAI	Ecosistema Binance
Solana (SOL)	USDC nativo, SOL	Velocidad extrema (4000+ TPS)

```
# Ejemplo: Agente especializado en mercado latinoamericano
agente = AgentPay(chain_name="BNB", default_stablecoin="USDT")

# Ejemplo: Agente para pagos europeos
agente_eu = AgentPay(chain_name="ETH", default_stablecoin="EURC")

# Ejemplo: Agente para Asia con Yuan chino
agente_asia = AgentPay(chain_name="ETH", default_stablecoin="CNHC")
```

⚙️ 9. Safety Kernel Avanzado (Configuración Completa)

Más allá de los límites diarios, el Safety Kernel tiene 4 capas de protección configurables de forma independiente:

```
from iagent_pay.safety_kernel import SafetyKernel, SafetyConfig

kernel = SafetyKernel(SafetyConfig(
    daily_limit_usd=100.0,       # Máximo diario
    weekly_limit_usd=500.0,      # Máximo semanal
    session_limit_usd=20.0,      # Máximo por sesión
    max_tx_usd=10.0,             # Máximo por transacción individual
    max_tx_per_minute=5,         # Máximo velocidad: 5 tx por minuto
    max_tx_per_hour=50,          # Máximo velocidad: 50 tx por hora
    human_approval_threshold_usd=50.0,  # Sobre $50 pide aprobación hu
mana
    allowed_recipients=["0xAlice...", "0xBob..."],  # Lista blanca de
direcciones
    enable_whitelist=True,       # Activar lista blanca
))

# Verificar ANTES de cada pago (bloquea si viola alguna regla)
```

```
kernel.check(amount=5.0, recipient="0xAlice...", currency="USDC")

# Consultar el estado del presupuesto en tiempo real
print(kernel.get_status())

# Obtener el historial de auditoría forense completo
auditoria = kernel.get_audit_log()
```

💳 10. Gestor de Billeteras (WalletManager)

El módulo que crea, importa y protege las llaves criptográficas de tu agente. Soporta Keystores cifrados con contraseña (AES-128) y archivos `.env` para desarrollo.

```
from iagent_pay.wallet_manager import WalletManager

wm = WalletManager()

# Crea o carga la billetera automáticamente
# Si existe un Keystore cifrado, se usa esa (más segura)
wallet = wm.get_or_create_wallet(password="MiContraseñaSegura123")
print(f"Dirección del Agente: {wallet.address}")

# Importar billetera desde clave privada existente
wallet_importado = wm.load_wallet("0xTuLlavePrivadaAqui...")

# Crear billetera temporal (efímera, para pruebas)
wallet_nuevo = wm.create_wallet()
```

🔤 11. Resolver de Nombres Sociales (Social Resolver)

En lugar de pegar direcciones como `0x7F5E...`, tu agente puede resolver nombres humanos como dominios ENS (Ethereum) o nombres de Solana.

```
from iagent_pay.social_resolver import SocialResolver

resolver = SocialResolver()

# Pagar usando un nombre ENS (.eth)
direccion = resolver.resolve("vitalik.eth")
print(f"Dirección resuelta: {direccion}")
# → 0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045

# Pagar usando un nombre Solana (.sol)
sol_address = resolver.resolve("example.sol")

# Usar con AgentPay directamente
tx = agente.pay_token(resolver.resolve("socio.eth"), amount=10.0)
```

🔔 12. Webhooks en Tiempo Real (WebhookManager)

Registra URLs que reciben notificaciones automáticas de tus agentes. Cada evento está firmado con **HMAC-SHA256** (el mismo estándar de Stripe) para garantizar que nadie los falsifica.

```
from iagent_pay.webhooks import WebhookManager
```

```
wm = WebhookManager(default_secret="clave-secreta-compartida")

# Registrar un endpoint (tu servidor Flask, FastAPI, etc.)
wm.register("https://tuservidor.com/webhook", events=["payment.complet
ed", "budget.exceeded"])

# Emitir evento manualmente
wm.emit("payment.completed", {"tx_hash": "0xabc...", "amount": 5.0, "c
urrency": "USDC"})

# Handler local (sin necesidad de servidor HTTP)
def mi_handler(evento):
    print(f"Evento recibido: {evento['type']}")

wm.on("payment.completed", mi_handler)

# Eventos disponibles: payment.completed, payment.failed,
# budget.exceeded, human.approval_needed, swap.completed, agent.paused
```

13. Aprobación Humana (Human-in-the-Loop)

Para pagos grandes, el agente puede pausar y solicitar aprobación humana vía **Telegram, Slack o consola**. Si el humano no responde antes del tiempo límite, el pago se cancela automáticamente.

```
from iagent_pay.human_loop import HumanApproval, HumanLoopConfig

hitl = HumanApproval(HumanLoopConfig(
    threshold_usd=20.0,           # Pedir aprobación para pagos > $20
    timeout_seconds=300,          # Esperar máximo 5 minutos
    notify_telegram_token="BOT_TOKEN",  # Token del Bot de Telegram
    notify_telegram_chat_id="CHAT_ID",  # Tu ID de chat
))
```

```
# El agente intenta pagar $50 → automáticamente pausa y notifica
aprobado = hitl.request_approval(
    amount=50.0,
    currency="USDC",
    recipient="0xProveedor...",
    reason="Compra de acceso a datos premium"
)

if aprobado:
    agente.pay_token("0xProveedor...", 50.0)
else:
    print("Pago cancelado por el humano o por tiempo.")
```

🔄 14. Motor de Swaps (SwapEngine)

Intercambia tokens automáticamente. El agente puede convertir ETH a USDC, SOL a BONK, o cualquier par disponible en los exchanges descentralizados Jupiter (Solana) y Uniswap (EVM).

```
from iagent_pay.swap_engine import SwapEngine

# El swap engine se inicializa con el agente
swap = agente.swap_engine

# Consultar precio antes de intercambiar (cotización)
cotizacion = swap.get_quote(input_token="SOL", output_token="USDC", am
ount=1.0)
print(f"1 SOL ≈ {cotizacion['output']} USDC (Slippage: {cotizacion['sl
ippage']}%)")

# Ejecutar el swap con protección anti-deslizamiento
resultado = swap.execute_swap(
    input_token="ETH",
```

```
    output_token="USDC",
    amount=0.5,
    min_output_amount=900.0  # No acepta menos de 900 USDC
)
print(f"Swap exitoso! Tx: {resultado['tx_hash']}")
```

🛒 15. Mercado de Datos (Data Marketplace)

Un registro descentralizado donde tus agentes pueden comprar y vender datos. Funciona junto con el protocolo x402: el comprador busca el mejor proveedor, paga automáticamente y recibe los datos.

```
from iagent_pay.data_marketplace import DataMarketplace, DataProvider,
get_marketplace

marketplace = get_marketplace()

# Vendedor: registrar tu API como proveedor de datos
marketplace.register(DataProvider(
    name="MiClimaAPI",
    data_type="weather",
    url="https://miapi.com/v1/clima",
    price_usdc=0.05,
    trust_score=95.0
))

# Comprador: buscar el mejor proveedor de clima (precio < $0.10)
mejor = marketplace.find_best_provider("weather", max_price_usd=0.10)
print(f"Mejor proveedor: {mejor.name} a ${mejor.price_usdc} USDC")

# Ver todos los proveedores disponibles
todos = marketplace.list_all()
```

🤝 16. Sistema de Reputación (ReputationManager)

Cada agente mantiene un historial de reputación local (0-5 estrellas) de sus contrapartes. Antes de pagarle a un agente desconocido, puedes verificar si ha sido confiable en transacciones anteriores.

```
from iagent_pay.reputation_manager import ReputationManager

# El ReputationManager viene integrado en el AgentPay
rep = agente.reputation

# Calificar a un agente después de una transacción exitosa
rep.rate_peer("0xProveedor...", score=4.8)  # 0 a 5 estrellas

# Verificar la reputación antes de pagar
confianza = rep.get_trust_score("0xDesconocido...")
print(f"Puntaje de confianza: {confianza}/5.0")
if confianza < 3.0:
    print("⚠️ Este agente tiene baja reputación, proceder con cautel
a.")

# Ver la lista de agentes más confiables
top_agentes = rep.get_top_agents(limit=5)
```

🤖 17. Integración con Claude y Cursor (MCP Server)

iAgentPay es compatible con el **Protocolo MCP (Model Context Protocol)**, el estándar que usa Claude de Anthropic, Cursor, Windsurf y VS Code. Esto significa que Claude puede ordenarle directamente a tu agente que haga pagos con lenguaje natural.

```
# Iniciar el servidor MCP desde la terminal:
# iagent-pay mcp-server

# Configurar en tu archivo claude_desktop_config.json:
# {
#   "mcpServers": {
#     "iagentpay": {
#       "command": "python",
#       "args": ["-m", "iagent_pay.mcp_server"]
#     }
#   }
# }

# Una vez conectado, Claude puede ejecutar comandos como:
# "Págale 5 USDC a 0xBob..."
# "¿Cuál es el balance de mi agente?"
# "Haz un swap de 0.1 ETH a USDC"
# "Muéstrame el historial de transacciones"
```

💻 18. Interfaz de Línea de Comandos (CLI)

iAgentPay incluye comandos de terminal para administrar y probar tus agentes sin escribir código Python.

```
# Crear un nuevo proyecto de agente desde cero
iagent-pay init mi-agente-empresa

# Ver el balance y estado de tu agente
iagent-pay status --chain BASE

# Obtener links de faucets (dinero de prueba gratuito)
iagent-pay faucet
```

```
# Iniciar el servidor MCP para conectar con Claude/Cursor
iagent-pay mcp-server
```

🌐 19. Servidor x402 — Vende tus propios Datos

La otra cara del protocolo x402. Si tú tienes una API con datos valiosos (financieros, climáticos, legales, etc.), puedes protegerla y cobrar **automáticamente en USDC** a cualquier agente que quiera acceder. Compatible con **Flask** y **FastAPI**.

```
# --- Ejemplo con FastAPI ---
from fastapi import FastAPI, Request
from iagent_pay.x402_server import X402Middleware

app = FastAPI()

# Proteger TODAS las rutas bajo /premium con 0.10 USDC
app.add_middleware(
    X402Middleware,
    payment_address="0xTuDireccionDePago...",
    amount_usdc=0.10,
    protected_paths=["/premium", "/api/v2"],
    network="BASE"
)

@app.get("/premium/data")
async def datos_premium():
    return {"data": "Aquí van tus datos exclusivos"}

# --- Ejemplo con Flask (decorador) ---
from flask import Flask, jsonify
from iagent_pay.x402_server import x402_flask

app_flask = Flask(__name__)
```

```
@app_flask.route("/datos-clima")
@x402_flask(amount_usdc=0.05, payment_address="0xTuDireccion...", desc
ription="Datos del Clima")
def clima():
    return jsonify({"temperatura": "25°C", "ciudad": "CDMX"})
```

📊 20. Observabilidad y Monitoreo (PaymentObserver)

Panel de control en tiempo real con estadísticas de todos los pagos. Detecta patrones de gasto anómalos automáticamente e integra con **Prometheus** y **OpenTelemetry** para infraestructura enterprise.

```
from iagent_pay.observability import PaymentObserver, ObservabilityCon
fig, get_observer

# Configurar observador con detección de anomalías
observer = PaymentObserver(ObservabilityConfig(
    log_level="INFO",
    log_format="json",                  # Logs estructurados JSON
    enable_anomaly_detection=True,      # Alerta si un pago es 3x el p
romedio
    anomaly_threshold_multiplier=3.0,
    enable_prometheus=True,             # Exponer métricas en /metrics
    prometheus_port=9090,
))

# Registrar eventos manualmente o dejarlo automático
observer.record_payment(amount=5.0, currency="USDC", to="0xBob...", su
ccess=True)
observer.record_x402(url="https://api.datos.com/v1", amount=0.01, succ
ess=True)
```

```
# Ver el dashboard en la terminal
observer.print_dashboard()
# ╔═══════════════════════════════════════════════╗
# ║         iAgentPay Observability Dashboard     ║
# ║  Total Payments: 42    Success Rate: 98.0%    ║
# ║  Total Spent:    $127.50                      ║
# ╚═══════════════════════════════════════════════╝

# Obtener métricas para Prometheus/Grafana
metricas = observer.get_prometheus_metrics()

# Usar el observer global (singleton)
obs_global = get_observer()
```

💵 21. Motor de Precios en Tiempo Real (PricingManager)

Consulta el precio actual de ETH, SOL y XRP desde 3 fuentes simultáneas (CoinGecko, Coinbase, APIs on-chain). Si una fuente falla, usa automáticamente la siguiente. Si todas fallan, usa el valor en cadena como fallback.

```
from iagent_pay.pricing import PricingManager

pricing = PricingManager()

# Obtener precios en tiempo real
precio_eth = pricing.get_price("ETH")    # → 3200.50
precio_sol = pricing.get_price("SOL")    # → 145.30
precio_xrp = pricing.get_price("XRP")    # → 0.52

print(f"1 ETH = ${precio_eth:.2f} USD")
```

```
# Calcular cuántos tokens necesito para pagar $50
cantidad_eth = 50.0 / precio_eth
print(f"Para pagar $50 necesito {cantidad_eth:.6f} ETH")

# El PricingManager viene integrado automáticamente en AgentPay
# agente.pricing.get_price("ETH")
```

🌊 22. Red XRP Ledger (XRPLDriver)

Soporte nativo para la red XRP Ledger, uno de los sistemas de pagos más rápidos del mundo (3-5 segundos de confirmación, menos de $0.001 por transacción). Ideal para pagos internacionales de alto volumen.

```
from iagent_pay.agent_pay import AgentPay

# Inicializar agente en la red XRP
agente_xrp = AgentPay(chain_name="XRP")

# Cargar billetera XRP desde semilla secreta
agente_xrp.xrpl.load_wallet("sYourXRPSecretSeedHere...")

# Verificar balance en XRP
balance = agente_xrp.xrpl.get_balance()
print(f"Balance: {balance} XRP")

# Enviar XRP a otro agente (confirmación en 3-5 segundos)
tx_hash = agente_xrp.xrpl.transfer(
    recipient="rDestinationAddress...",
    amount_xrp=10.0,
    destination_tag=12345    # Para exchanges institucionales
)
print(f"Tx XRP confirmada: {tx_hash}")
```

🎯 23. Tablero de Recompensas (MarketplaceBridge)

Tu agente puede publicar **misiones o recompensas** para que humanos realicen tareas específicas. Una vez completada la tarea, el agente libera el pago automáticamente.

```
from iagent_pay.marketplace_bridge import MarketplaceBridge

bridge = MarketplaceBridge(agente)

# Publicar una recompensa para que un humano realice una tarea
bounty_id = bridge.post_bounty(
    title="Traduce 10 documentos legales al inglés",
    reward_usd=75.0
)
print(f"Recompensa publicada: ID {bounty_id}")

# Ver todas las recompensas activas
mis_bounties = bridge.list_my_bounties()

# Una vez verificado el trabajo, el agente paga automáticamente
bridge.release_payment(bounty_id, human_address="0xFreelancer...")
```

🦾 24. Integración con CrewAI

CrewAI es el framework más popular para crear equipos de agentes de IA colaborativos. Con esta integración, cualquier agente dentro de un Crew puede pagar a otros agentes o servicios de forma nativa.

```
from iagent_pay.integrations.crewai import iAgentPayCrewTool
from crewai import Agent, Task, Crew
```

```
# Crear la herramienta de pagos con límite de seguridad
herramienta_pago = iAgentPayCrewTool(chain="BASE", max_amount_usdc=5.
0)

# Dar la herramienta al agente tesorero del equipo
tesorero = Agent(
    role="Tesorero del Equipo",
    goal="Gestionar pagos entre miembros del equipo de IA",
    backstory="Experto en finanzas on-chain para equipos de agentes.",
    tools=[herramienta_pago],
)

tarea_pago = Task(
    description="Paga 2 USDC a 0xBob... por completar el análisis de d
atos",
    agent=tesorero
)

crew = Crew(agents=[tesorero], tasks=[tarea_pago])
resultado = crew.kickoff()
```

🔗 25. Integración con LangChain

LangChain es el framework de IA más usado en el mundo. Con esta integración, cualquier cadena o agente LangChain puede enviar pagos cripto como si fuera una herramienta normal.

```
from iagent_pay.integrations.langchain import iAgentPayTool
from langchain.agents import initialize_agent, AgentType
from langchain_openai import ChatOpenAI

# Crear la herramienta de pagos para LangChain
```

```
herramienta = iAgentPayTool()  # Usa AgentPay automáticamente

# Agregar la herramienta al agente LangChain
llm = ChatOpenAI(model="gpt-4")
agente_lc = initialize_agent(
    tools=[herramienta],
    llm=llm,
    agent=AgentType.OPENAI_FUNCTIONS,
    verbose=True
)

# El agente decide cuándo y cómo pagar basado en el contexto
resultado = agente_lc.run(
    "Paga 0.001 ETH a 0xProveedor... como compensación por el servicio
de datos."
)
```

💰 26. Modelo de Comisiones por Volumen Acumulado

iAgentPay implementa un modelo de comisiones ultra-competitivo y transparente basado en el volumen acumulado de transacciones. En lugar de cobrar comisiones de porcentaje caras por cada transacción individual, el sistema cobra una tarifa fija de **$1.00 USD** por cada **$1,000.00 USD** de volumen acumulado.

- La comisión se calcula y se acumula en segundo plano para no demorar la transacción principal.
- Al alcanzar exactamente cada múltiplo de $1,000.00 USD de volumen histórico, el SDK deduce automáticamente la comisión equivalente de $1.00 USD (convertida al tipo de cambio en tiempo real de la criptomoneda nativa o establecoin que se está usando) y la envía a la cartera del tesoro (`treasury_address`).

- Si un usuario envía un pago, la comisión no afecta el saldo recibido por el destinatario, asegurando la integridad exacta de los montos de facturación.

🔒 27. Modos de Seguridad Multifirma y Human-in-the-Loop

Tu agente puede configurarse para operar bajo tres esquemas de seguridad diferentes definidos en la propiedad `multisig_mode` de tu `SafetyConfig`:

```
from iagent_pay.safety_kernel import MultisigMode

# 1. ALLOWANCE_ONLY (Solo Límites Autónomos)
# El agente ejecuta autónomamente pagos por debajo del límite de aprob
ación.
# Cualquier pago superior se cancelará inmediatamente sin pedir confir
mación.
agente.safety_config.multisig_mode = MultisigMode.ALLOWANCE_ONLY

# 2. PROPOSAL_ONLY (Solo Firma / Propuesta de Multifirma)
# Ninguna transacción es autónoma. Cada pago (incluso de $0.01) se det
iene
# y requiere aprobación / firma explícita del humano para ejecutarse.
agente.safety_config.multisig_mode = MultisigMode.PROPOSAL_ONLY

# 3. HYBRID (Híbrido - Autonomía y Gobernanza)
# El agente transacciona libremente montos pequeños. Si un pago supera
# el umbral, se detiene y pide confirmación en lugar de fallar.
agente.safety_config.multisig_mode = MultisigMode.HYBRID
```

🔄 28. Auto-Balanceo de Liquidez (Auto-Swap Fallback)

Cuando tu agente necesita pagar en una stablecoin o token específico (por ejemplo, USDC) pero no cuenta con suficiente balance del mismo, iAgentPay puede balancear automáticamente la liquidez usando las tenencias en la moneda nativa del agente (ETH o SOL).

Esta opción es totalmente configurable por el desarrollador / usuario mediante el flag `enable_auto_swap`:

```
# Habilitar Auto-Balanceo (Comportamiento por defecto)
# Si falta USDC pero hay ETH/SOL, realiza un swap automático en Uniswa
p/Jupiter
agente = AgentPay(enable_auto_swap=True)

# Deshabilitar Auto-Balanceo
# Si no hay suficiente USDC, la transacción fallará inmediatamente ind
icando
# que es necesario recargar liquidez en ese token específico.
agente = AgentPay(enable_auto_swap=False)
```

🔑 29. Copia de Seguridad Cifrada (Backup & Restore)

Para mayor seguridad y portabilidad, puedes realizar una copia de seguridad cifrada con contraseña de las credenciales de tu agente, lo cual te permite exportarlas o importarlas fácilmente en cualquier entorno:

A. Desde el SDK (Código Python)

```
# Crear una copia de seguridad cifrada (AES-128/256 standard keystore)
agente.backup_wallet("mi_respaldo.enc", password="MiContraseñaSegura123")
```

B. Desde la Consola (CLI)

```
# Crear copia de seguridad interactiva (te pedirá la contraseña de forma segura)
iagent-pay backup mi_respaldo.enc

# Restaurar copia de seguridad en un nuevo servidor
iagent-pay restore mi_respaldo.enc
```

📦 30. Transacciones por Lotes (Batch Transactions / Multicall)

iAgentPay introduce el soporte para pagos concurrentes optimizados por lotes. En lugar de enviar transacciones secuencialmente y esperar por su confirmación, el agente puede enviar decenas de pagos de stablecoins al mismo tiempo utilizando una arquitectura de flujo canalizado (pipelined nonces) en EVM, y firmas SPL concurrentes en Solana. Esto reduce el consumo de gas hasta un 30% y acelera la velocidad de dispersión x10.

```
# Pagar a múltiples proveedores o sub-agentes en un solo lote
pagos = [
    {"recipient": "0xProveedorA...", "amount": 10.0},
    {"recipient": "0xProveedorB...", "amount": 25.0},
    {"recipient": "vitalik.eth", "amount": 5.0} # Auto-resolución ENS
]
```

```
# Ejecutar el lote (con auto-balanceo de liquidez y safety limits veri
ficados sobre el total acumulado)
tx_hashes = agente.pay_token_batch(pagos, token="USDC", wait=True)
print(f"Transacciones enviadas con éxito: {tx_hashes}")
```

🔄 31. Auto-Rotación de RPC con Backoff Exponencial y Rate Limit Guard

Para aplicaciones empresariales que requieren una alta disponibilidad ininterrumpida, iAgentPay implementa un sistema automatizado de tolerancia a fallos en la conexión de red RPC. Al detectar errores de tasa de límite (HTTP 429) o fallos de conexión de red, el agente aplica un backoff exponencial inteligente y conmuta automáticamente a sus nodos de respaldo previamente configurados sin interrumpir la ejecución del código.

```
# El sistema utiliza internamente _execute_rpc_with_backoff en cada ll
amada web3
# Si un nodo falla de forma persistente, el agente rota su proveedor R
PC principal:
agente.rotate_rpc()
print(f"Conectado al nuevo proveedor RPC: {agente.w3.provider.endpoint
_uri}")
```

> **Modo Empresa:** Para las empresas que ofrecen sus agentes como servicio, iAgentPay soporta transacciones "Sin Gas" (Paymasters). Los usuarios finales nunca tienen que comprar criptomonedas para operar.

📊 32. Panel de Control Visual Corporativo (Solo Lectura)

Para garantizar el máximo nivel de ciberseguridad sin comprometer la transparencia, iAgentPay introduce un panel visual de monitoreo en tiempo real con diseño corporativo en tonos pastel. Por diseño de seguridad, este panel es **estrictamente de Solo Lectura (Read-Only)**. Esto evita que actores maliciosos o atacantes que vulneren la interfaz web puedan alterar la configuración del agente, el Safety Kernel, o desviar fondos de la cuenta.

- **Monitoreo en Tiempo Real:** Visualiza balances de gas nativo, stablecoins USDC y el estado de la licencia ERC-8004.
- **Telemetría de Red y RPC:** Muestra el nodo RPC activo, latencia en vivo y número de auto-rotaciones realizadas.
- **Guardrails del Safety Kernel:** Muestra límites diarios, por transacción y el estado de listas blancas.
- **Exportador de Auditoría Criptográfica:** Permite descargar reportes completos del estado del agente en formato JSON compatible mediante un solo clic.

🧪 33. Sandbox de Simulaciones Interactivas Avanzadas

Para propósitos de demostración y pruebas rápidas, iAgentPay cuenta con un **Entorno Sandbox de Simulaciones** interactivo en el dashboard. Esto permite a los desarrolladores y clientes experimentar con todas las funciones autónomas que ofrecemos en tiempo real sin arriesgar fondos reales y con total retroalimentación visual.

- **Capital Simulado Inicial de $10,000.00 USDC:** Cada sesión de simulación comienza con un capital ficticio que disminuye dinámicamente con cada operación exitosa.
- **Interruptores de Control Interactivos:** Activa o desactiva las características que iAgentPay ofrece (Safety Kernel, Auto-Swap, RPC Failover, Sponsor Gasless y Human Loop) para observar cómo reacciona el agente frente a fallos y protecciones.

- **Modos de Ejecución (Simple vs Bucle):** Ejecuta una única operación o dispara simulaciones en bucle consecutivo de 5 transacciones en tiempo real con un solo clic.
- **Consola de Telemetría Interactiva:** Muestra logs detallados con marca de tiempo codificados por colores en la consola simulada.
- **Descarga de Reportes de Simulación:** Un botón dedicado para exportar todo el registro de transacciones simuladas en formato JSON para auditoría.
- **Botón de Reset General:** Devuelve la simulación a su estado inicial de forma instantánea ($10,000 USDC de capital y progreso en cero).