WayBank v6.0 - Sistema de Gestión de Liquidez Concentrada en Uniswap V3

Tutorial Completo para waypool.net

Versión: 6.0
Fecha: Enero 2026
Autor: Elysium Media FZCO

📋 Índice

---

1. Visión General del Sistema

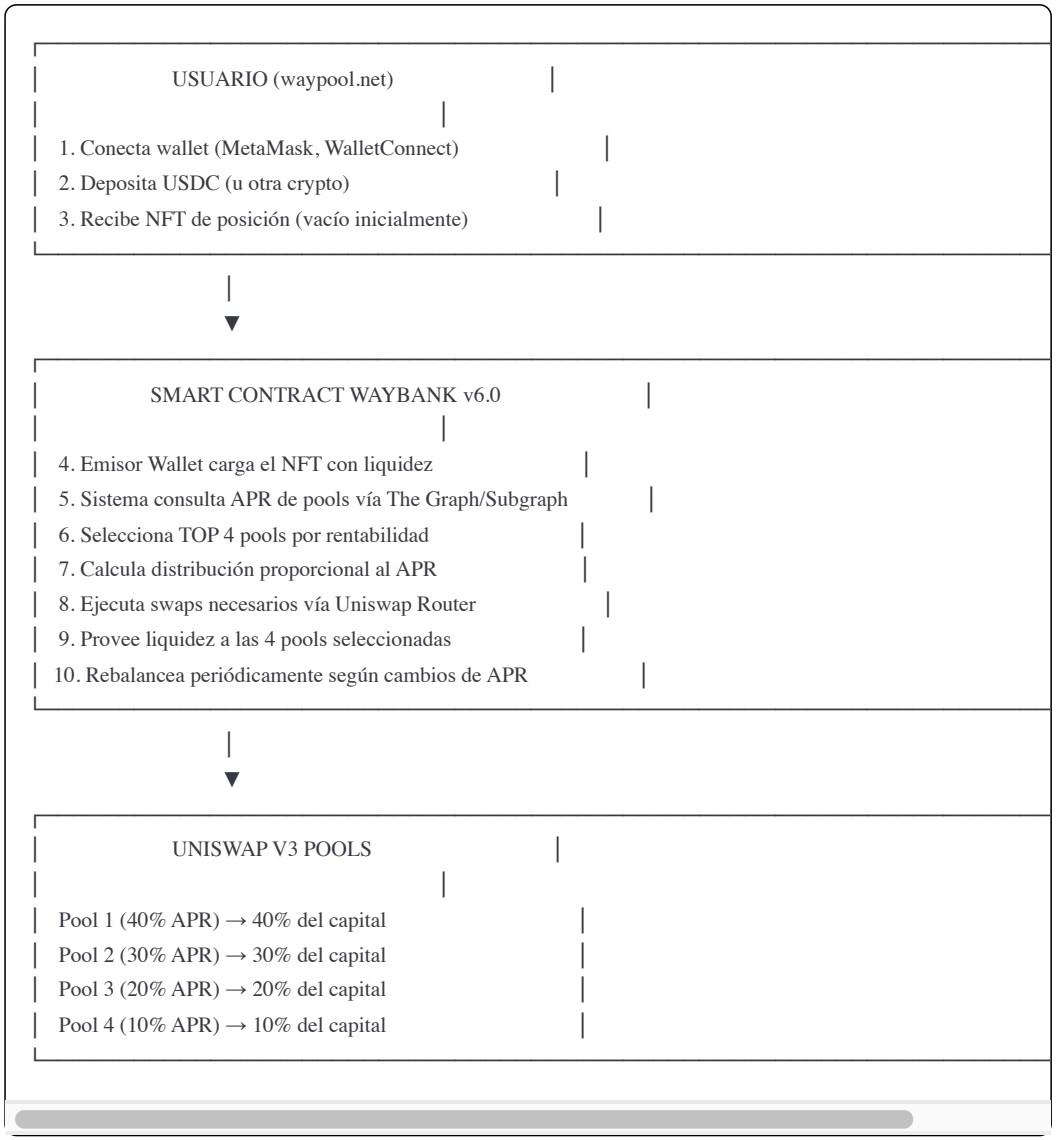
1.1 ¿Qué es WayBank v6.0?

WayBank v6.0 es un smart contract de gestión automatizada de liquidez concentrada que:

- **Monitoriza** las 4 pools más rentables de Uniswap V3
- **Distribuye** capital proporcionalmente según el APR de cada pool

- **Rebalancea** automáticamente cuando cambian las condiciones del mercado
- **Ejecuta swaps** para convertir los tokens necesarios para cada pool
- **Gestiona NFTs** de posición de Uniswap V3 para los usuarios

1.2 Flujo Principal



1.3 Componentes Principales

Componente	Descripción
WayBankVault.sol	Contrato principal que gestiona depósitos y NFTs
PoolAnalyzer.sol	Analiza y rankea pools por APR
LiquidityManager.sol	Gestiona posiciones en Uniswap V3
SwapExecutor.sol	Ejecuta swaps para preparar tokens
Rebalancer.sol	Lógica de rebalanceo automático
Backend API	Servidor Node.js para datos off-chain
Frontend	Interfaz React en waypool.net

---

2. Arquitectura del Sistema

2.1 Estructura de Contratos

```
waybank-v6/
├── contracts/
│   ├── core/
│   │   ├── WayBankVault.sol        # Contrato principal
│   │   ├── WayBankStorage.sol      # Storage separado (upgradeable pattern)
│   │   └── WayBankFactory.sol      # Factory para crear vaults
│   ├── modules/
│   │   ├── PoolAnalyzer.sol        # Análisis de pools
│   │   ├── LiquidityManager.sol    # Gestión de liquidez
│   │   ├── SwapExecutor.sol        # Ejecución de swaps
│   │   └── Rebalancer.sol          # Sistema de rebalanceo
│   ├── interfaces/
│   │   ├── IWayBankVault.sol
│   │   ├── IUniswapV3Pool.sol
│   │   ├── INonfungiblePositionManager.sol
│   │   └── ISwapRouter.sol
│   └── libraries/
│       ├── TickMath.sol
```

```
│       ├── LiquidityAmounts.sol
│       └── PoolAddress.sol
├── scripts/
│   ├── deploy.js
│   ├── verify.js
│   └── rebalance-cron.js
├── test/
│   └── WayBank.test.js
└── hardhat.config.js
```

2.2 Direcciones de Contratos Uniswap V3 (Mainnet)

javascript

```
// Ethereum Mainnet
const ADDRESSES = {
  // Uniswap V3 Core
  FACTORY: "0x1F98431c8aD98523631AE4a59f267346ea31F984",

  // Uniswap V3 Periphery
  NONFUNGIBLE_POSITION_MANAGER: "0xC36442b4a4522E871399CD717aBDD847Ab11FE88",
  SWAP_ROUTER: "0xE592427A0AEce92De3Edee1F18E0157C05861564",
  SWAP_ROUTER_02: "0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45",
  QUOTER: "0xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6",
  QUOTER_V2: "0x61fFE014bA17989E743c5F6cB21bF9697530B21e",

  // Tokens comunes
  WETH: "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2",
  USDC: "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48",
  USDT: "0xdAC17F958D2ee523a2206206994597C13D831ec7",
  DAI: "0x6B175474E89094C44Da98b954EedeAC495271d0F",
  WBTC: "0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599"
};

// Polygon Mainnet
const POLYGON_ADDRESSES = {
  FACTORY: "0x1F98431c8aD98523631AE4a59f267346ea31F984",
  NONFUNGIBLE_POSITION_MANAGER: "0xC36442b4a4522E871399CD717aBDD847Ab11FE88",
  SWAP_ROUTER_02: "0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45",
  WMATIC: "0x0d500B1d8E8eF31E21C99d1Db9A6444d3ADf1270",
  USDC: "0x2791Bca1f2de4661ED88A30C99A7a9449Aa84174"
};

// Arbitrum One
const ARBITRUM_ADDRESSES = {
  FACTORY: "0x1F98431c8aD98523631AE4a59f267346ea31F984",
  NONFUNGIBLE_POSITION_MANAGER: "0xC36442b4a4522E871399CD717aBDD847Ab11FE88",
  SWAP_ROUTER_02: "0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45"
};
```

2.3 Fee Tiers de Uniswap V3

javascript

```
const FEE_TIERS = {
  LOWEST: 100,   // 0.01% - Pares muy estables (stablecoins)
  LOW: 500,      // 0.05% - Pares estables
  MEDIUM: 3000,  // 0.30% - Pares estándar (más común)
  HIGH: 10000    // 1.00% - Pares exóticos/volátiles
};

const TICK_SPACINGS = {
  100: 1,        // Fee 0.01%
  500: 10,       // Fee 0.05%
  3000: 60,      // Fee 0.30%
  10000: 200     // Fee 1.00%
};
```

---

3. Contratos Inteligentes Necesarios

3.1 Contrato Principal: WayBankVault.sol

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@uniswap/v3-periphery/contracts/interfaces/INonfungiblePositionManager.sol";
import "@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol";
import "@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol";
import "@uniswap/v3-core/contracts/interfaces/IUniswapV3Factory.sol";
import "@uniswap/v3-core/contracts/libraries/TickMath.sol";

/**
 * @title WayBankVault
 * @notice Gestiona liquidez concentrada en Uniswap V3 distribuyendo capital
 *         entre las 4 pools más rentables según APR
 * @dev Implementa IERC721Receiver para recibir NFTs de posición
 */
contract WayBankVault is IERC721Receiver, ReentrancyGuard, Ownable {
    using SafeERC20 for IERC20;

    // ============ ESTRUCTURAS ============

    /// @notice Información de un depósito de usuario
    struct UserDeposit {
        uint256 amount;          // Cantidad depositada en USDC
        uint256 depositTime;     // Timestamp del depósito
        uint256[] positionIds;   // IDs de NFTs de posición asignados
        bool isActive;           // Si el depósito está activo
    }

    /// @notice Información de una pool seleccionada
    struct PoolInfo {
        address poolAddress;     // Dirección de la pool
        address token0;          // Token 0
        address token1;          // Token 1
        uint24 fee;              // Fee tier
        uint256 apr;             // APR estimado (en basis points, 10000 = 100%)
        uint256 allocatedAmount; // Cantidad asignada a esta pool
```

```
    int24 tickLower;        // Tick inferior de la posición
    int24 tickUpper;        // Tick superior de la posición
}

/// @notice Información de una posición NFT
struct PositionInfo {
    uint256 tokenId;        // ID del NFT
    address owner;          // Propietario original
    address pool;           // Pool de la posición
    uint128 liquidity;      // Liquidez
    int24 tickLower;
    int24 tickUpper;
}

// ============ CONSTANTES ============

uint256 public constant MAX_POOLS = 4;
uint256 public constant BASIS_POINTS = 10000;
uint256 public constant MIN_DEPOSIT = 10 * 1e6; // 10 USDC mínimo

// ============ INMUTABLES ============

INonfungiblePositionManager public immutable positionManager;
ISwapRouter public immutable swapRouter;
IUniswapV3Factory public immutable factory;
IERC20 public immutable depositToken; // USDC

// ============ ESTADO ============

/// @notice Wallet emisor que carga las posiciones
address public emitterWallet;

/// @notice Pools actualmente seleccionadas (máximo 4)
PoolInfo[MAX_POOLS] public selectedPools;
uint256 public activePoolCount;

/// @notice Mapeo de usuario a su depósito
mapping(address => UserDeposit) public userDeposits;

/// @notice Mapeo de tokenId de NFT a su información
mapping(uint256 => PositionInfo) public positions;
```

```
    /// @notice Total depositado en el vault
    uint256 public totalDeposited;

    /// @notice Última vez que se rebalanceó
    uint256 public lastRebalanceTime;

    /// @notice Intervalo mínimo entre rebalanceos (default: 1 hora)
    uint256 public rebalanceInterval = 1 hours;

    // ============ EVENTOS ============

    event Deposited(address indexed user, uint256 amount, uint256 timestamp);
    event Withdrawn(address indexed user, uint256 amount, uint256 timestamp);
    event NFTMinted(address indexed user, uint256 tokenId, address pool);
    event PoolsUpdated(address[4] pools, uint256[4] aprs);
    event Rebalanced(uint256 timestamp, uint256[4] newAllocations);
    event EmitterUpdated(address oldEmitter, address newEmitter);
    event FeesCollected(uint256 tokenId, uint256 amount0, uint256 amount1);

    // ============ MODIFICADORES ============

    modifier onlyEmitter() {
        require(msg.sender == emitterWallet, "WayBank: not emitter");
        _;
    }

    // ============ CONSTRUCTOR ============

    /**
     * @param _positionManager Dirección del NonfungiblePositionManager de Uniswap V3
     * @param _swapRouter Dirección del SwapRouter de Uniswap V3
     * @param _factory Dirección del Factory de Uniswap V3
     * @param _depositToken Token de depósito (USDC)
     * @param _emitterWallet Wallet que cargará las posiciones
     */
    constructor(
        address _positionManager,
        address _swapRouter,
        address _factory,
        address _depositToken,
        address _emitterWallet
    ) {
```

```
        require(_positionManager != address(0), "Invalid position manager");
        require(_swapRouter != address(0), "Invalid swap router");
        require(_factory != address(0), "Invalid factory");
        require(_depositToken != address(0), "Invalid deposit token");
        require(_emitterWallet != address(0), "Invalid emitter wallet");

        positionManager = INonfungiblePositionManager(_positionManager);
        swapRouter = ISwapRouter(_swapRouter);
        factory = IUniswapV3Factory(_factory);
        depositToken = IERC20(_depositToken);
        emitterWallet = _emitterWallet;
    }

    // ============ FUNCIONES DE USUARIO ============

    /**
     * @notice Deposita tokens y recibe un NFT de posición vacío
     * @param amount Cantidad a depositar
     * @return tokenId ID del NFT de posición creado
     */
    function deposit(uint256 amount) external nonReentrant returns (uint256 tokenId) {
        require(amount >= MIN_DEPOSIT, "WayBank: amount too low");
        require(!userDeposits[msg.sender].isActive, "WayBank: deposit exists");

        // Transferir tokens del usuario al contrato
        depositToken.safeTransferFrom(msg.sender, address(this), amount);

        // Crear el registro de depósito
        userDeposits[msg.sender] = UserDeposit({
            amount: amount,
            depositTime: block.timestamp,
            positionIds: new uint256[](0),
            isActive: true
        });

        totalDeposited += amount;

        emit Deposited(msg.sender, amount, block.timestamp);

        // El NFT será minteado y cargado por el emitter wallet
        // Retornamos 0 indicando que el NFT aún no existe
        return 0;
```

```
}

/**
 * @notice Retira el depósito y los fees acumulados
 */
function withdraw() external nonReentrant {
    UserDeposit storage userDep = userDeposits[msg.sender];
    require(userDep.isActive, "WayBank: no active deposit");

    uint256 totalToReturn = userDep.amount;

    // Recolectar fees de todas las posiciones del usuario
    for (uint256 i = 0; i < userDep.positionIds.length; i++) {
        uint256 tokenId = userDep.positionIds[i];
        (uint256 fees0, uint256 fees1) = _collectFees(tokenId);
        // Los fees se convertirán a USDC y se añadirán al total
    }

    // Remover liquidez de las posiciones
    for (uint256 i = 0; i < userDep.positionIds.length; i++) {
        uint256 tokenId = userDep.positionIds[i];
        _removeLiquidity(tokenId);
        // Quemar el NFT
        positionManager.burn(tokenId);
    }

    // Limpiar el estado del usuario
    totalDeposited -= userDep.amount;
    delete userDeposits[msg.sender];

    // Transferir tokens al usuario
    depositToken.safeTransfer(msg.sender, totalToReturn);

    emit Withdrawn(msg.sender, totalToReturn, block.timestamp);
}

// ============ FUNCIONES DEL EMITTER ============

/**
 * @notice El emitter wallet carga las posiciones del usuario
 * @param user Dirección del usuario
 * @param pools Array de direcciones de pools donde proveer liquidez
```

```
     * @param amounts Array de cantidades para cada pool
     */
    function loadUserPositions(
        address user,
        address[] calldata pools,
        uint256[] calldata amounts
    ) external onlyEmitter nonReentrant {
        require(pools.length <= MAX_POOLS, "WayBank: too many pools");
        require(pools.length == amounts.length, "WayBank: length mismatch");

        UserDeposit storage userDep = userDeposits[user];
        require(userDep.isActive, "WayBank: no active deposit");

        uint256 totalAllocated = 0;
        for (uint256 i = 0; i < amounts.length; i++) {
            totalAllocated += amounts[i];
        }
        require(totalAllocated <= userDep.amount, "WayBank: exceeds deposit");

        // Para cada pool, crear posición
        for (uint256 i = 0; i < pools.length; i++) {
            if (amounts[i] > 0) {
                uint256 tokenId = _mintPosition(user, pools[i], amounts[i]);
                userDep.positionIds.push(tokenId);

                emit NFTMinted(user, tokenId, pools[i]);
            }
        }
    }

    /**
     * @notice Actualiza las pools seleccionadas basándose en datos de APR
     * @param poolAddresses Direcciones de las 4 pools
     * @param aprs APR de cada pool (en basis points)
     */
    function updateSelectedPools(
        address[4] calldata poolAddresses,
        uint256[4] calldata aprs
    ) external onlyEmitter {
        // Calcular APR total para distribución proporcional
        uint256 totalApr = 0;
        for (uint256 i = 0; i < MAX_POOLS; i++) {
```

```
        totalApr += aprs[i];
    }
    require(totalApr > 0, "WayBank: total APR is zero");

    activePoolCount = 0;

    for (uint256 i = 0; i < MAX_POOLS; i++) {
        if (poolAddresses[i] != address(0)) {
            IUniswapV3Pool pool = IUniswapV3Pool(poolAddresses[i]);

            selectedPools[i] = PoolInfo({
                poolAddress: poolAddresses[i],
                token0: pool.token0(),
                token1: pool.token1(),
                fee: pool.fee(),
                apr: aprs[i],
                allocatedAmount: 0,
                tickLower: 0,
                tickUpper: 0
            });

            activePoolCount++;
        }
    }

    emit PoolsUpdated(poolAddresses, aprs);
}

/**
 * @notice Rebalancea las posiciones según los nuevos APRs
 */
function rebalance() external onlyEmitter nonReentrant {
    require(
        block.timestamp >= lastRebalanceTime + rebalanceInterval,
        "WayBank: too soon to rebalance"
    );

    // Calcular nueva distribución basada en APRs actuales
    uint256 totalApr = 0;
    for (uint256 i = 0; i < activePoolCount; i++) {
        totalApr += selectedPools[i].apr;
    }
```

```
    uint256[4] memory newAllocations;

    for (uint256 i = 0; i < activePoolCount; i++) {
        // Distribución proporcional al APR
        newAllocations[i] = (totalDeposited * selectedPools[i].apr) / totalApr;
        selectedPools[i].allocatedAmount = newAllocations[i];
    }

    lastRebalanceTime = block.timestamp;

    emit Rebalanced(block.timestamp, newAllocations);
}

// ============ FUNCIONES INTERNAS ============

/**
 * @notice Mintea una nueva posición de liquidez
 */
function _mintPosition(
    address user,
    address poolAddress,
    uint256 amount
) internal returns (uint256 tokenId) {
    IUniswapV3Pool pool = IUniswapV3Pool(poolAddress);

    address token0 = pool.token0();
    address token1 = pool.token1();
    uint24 fee = pool.fee();

    // Obtener el tick actual
    (uint160 sqrtPriceX96, int24 currentTick,,,,,) = pool.slot0();

    // Calcular tick range (±10% del precio actual)
    int24 tickSpacing = pool.tickSpacing();
    int24 tickLower = ((currentTick - 1000) / tickSpacing) * tickSpacing;
    int24 tickUpper = ((currentTick + 1000) / tickSpacing) * tickSpacing;

    // Calcular cuánto de cada token necesitamos
    // Esto es una simplificación - en producción usar LiquidityAmounts
    uint256 amount0 = amount / 2;
    uint256 amount1 = amount / 2;
```

```
    // Hacer swaps si es necesario para obtener los tokens
    _swapForTokens(token0, token1, amount0, amount1);

    // Aprobar tokens al position manager
    IERC20(token0).safeApprove(address(positionManager), amount0);
    IERC20(token1).safeApprove(address(positionManager), amount1);

    // Mintear la posición
    INonfungiblePositionManager.MintParams memory params =
        INonfungiblePositionManager.MintParams({
            token0: token0,
            token1: token1,
            fee: fee,
            tickLower: tickLower,
            tickUpper: tickUpper,
            amount0Desired: amount0,
            amount1Desired: amount1,
            amount0Min: 0,
            amount1Min: 0,
            recipient: address(this),
            deadline: block.timestamp + 15 minutes
        });

    (tokenId,,,) = positionManager.mint(params);

    // Registrar la posición
    positions[tokenId] = PositionInfo({
        tokenId: tokenId,
        owner: user,
        pool: poolAddress,
        liquidity: 0, // Se actualizará
        tickLower: tickLower,
        tickUpper: tickUpper
    });

    return tokenId;
}

/**
 * @notice Ejecuta swaps para obtener los tokens necesarios
 */
```

```
function _swapForTokens(
    address token0,
    address token1,
    uint256 amount0Needed,
    uint256 amount1Needed
) internal {
    address usdcAddress = address(depositToken);

    // Si token0 no es USDC, hacer swap
    if (token0 != usdcAddress && amount0Needed > 0) {
        _executeSwap(usdcAddress, token0, amount0Needed);
    }

    // Si token1 no es USDC, hacer swap
    if (token1 != usdcAddress && amount1Needed > 0) {
        _executeSwap(usdcAddress, token1, amount1Needed);
    }
}

/**
 * @notice Ejecuta un swap único
 */
function _executeSwap(
    address tokenIn,
    address tokenOut,
    uint256 amountIn
) internal returns (uint256 amountOut) {
    IERC20(tokenIn).safeApprove(address(swapRouter), amountIn);

    ISwapRouter.ExactInputSingleParams memory params =
        ISwapRouter.ExactInputSingleParams({
            tokenIn: tokenIn,
            tokenOut: tokenOut,
            fee: 3000, // 0.3% por defecto
            recipient: address(this),
            deadline: block.timestamp + 15 minutes,
            amountIn: amountIn,
            amountOutMinimum: 0, // En producción usar oracle
            sqrtPriceLimitX96: 0
        });

    amountOut = swapRouter.exactInputSingle(params);
```

```
}

/**
 * @notice Recolecta fees de una posición
 */
function _collectFees(uint256 tokenId) internal returns (uint256 amount0, uint256 amount1) {
    INonfungiblePositionManager.CollectParams memory params =
        INonfungiblePositionManager.CollectParams({
            tokenId: tokenId,
            recipient: address(this),
            amount0Max: type(uint128).max,
            amount1Max: type(uint128).max
        });

    (amount0, amount1) = positionManager.collect(params);

    emit FeesCollected(tokenId, amount0, amount1);
}

/**
 * @notice Remueve toda la liquidez de una posición
 */
function _removeLiquidity(uint256 tokenId) internal returns (uint256 amount0, uint256 amount1) {
    (,,,,,,,uint128 liquidity,,,,) = positionManager.positions(tokenId);

    if (liquidity > 0) {
        INonfungiblePositionManager.DecreaseLiquidityParams memory params =
            INonfungiblePositionManager.DecreaseLiquidityParams({
                tokenId: tokenId,
                liquidity: liquidity,
                amount0Min: 0,
                amount1Min: 0,
                deadline: block.timestamp + 15 minutes
            });

        (amount0, amount1) = positionManager.decreaseLiquidity(params);
    }

    // Recolectar los tokens
    _collectFees(tokenId);
}
```

```
// ============ FUNCIONES ADMIN ============

/**
 * @notice Actualiza la wallet del emitter
 */
function setEmitterWallet(address _newEmitter) external onlyOwner {
    require(_newEmitter != address(0), "WayBank: invalid emitter");
    address oldEmitter = emitterWallet;
    emitterWallet = _newEmitter;
    emit EmitterUpdated(oldEmitter, _newEmitter);
}

/**
 * @notice Actualiza el intervalo de rebalanceo
 */
function setRebalanceInterval(uint256 _interval) external onlyOwner {
    require(_interval >= 5 minutes, "WayBank: interval too short");
    rebalanceInterval = _interval;
}

// ============ FUNCIONES VIEW ============

/**
 * @notice Obtiene información del depósito de un usuario
 */
function getUserDeposit(address user) external view returns (
    uint256 amount,
    uint256 depositTime,
    uint256[] memory positionIds,
    bool isActive
) {
    UserDeposit storage dep = userDeposits[user];
    return (dep.amount, dep.depositTime, dep.positionIds, dep.isActive);
}

/**
 * @notice Obtiene las pools seleccionadas actualmente
 */
function getSelectedPools() external view returns (PoolInfo[4] memory) {
    return selectedPools;
}
```

```
    /**
     * @notice Calcula la distribución óptima para una cantidad dada
     */
    function calculateDistribution(uint256 amount) external view returns (
        uint256[4] memory allocations
    ) {
        uint256 totalApr = 0;
        for (uint256 i = 0; i < activePoolCount; i++) {
            totalApr += selectedPools[i].apr;
        }

        if (totalApr == 0) return allocations;

        for (uint256 i = 0; i < activePoolCount; i++) {
            allocations[i] = (amount * selectedPools[i].apr) / totalApr;
        }
    }

    // ============ ERC721 RECEIVER ============

    function onERC721Received(
        address,
        address,
        uint256,
        bytes calldata
    ) external pure override returns (bytes4) {
        return this.onERC721Received.selector;
    }
}
```

3.2 Contrato PoolAnalyzer.sol

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "@uniswap/v3-core/contracts/interfaces/IUniswapV3Pool.sol";
import "@uniswap/v3-core/contracts/interfaces/IUniswapV3Factory.sol";

/**
 * @title PoolAnalyzer
 * @notice Analiza pools de Uniswap V3 para obtener métricas on-chain
 */
contract PoolAnalyzer {

    IUniswapV3Factory public immutable factory;

    struct PoolMetrics {
        address poolAddress;
        address token0;
        address token1;
        uint24 fee;
        uint128 liquidity;
        uint160 sqrtPriceX96;
        int24 tick;
        uint256 feeGrowthGlobal0X128;
        uint256 feeGrowthGlobal1X128;
    }

    constructor(address _factory) {
        factory = IUniswapV3Factory(_factory);
    }

    /**
     * @notice Obtiene métricas de una pool específica
     */
    function getPoolMetrics(address poolAddress) external view returns (PoolMetrics memory) {
        IUniswapV3Pool pool = IUniswapV3Pool(poolAddress);

        (uint160 sqrtPriceX96, int24 tick,,,,,) = pool.slot0();

        return PoolMetrics({
            poolAddress: poolAddress,
            token0: pool.token0(),
```

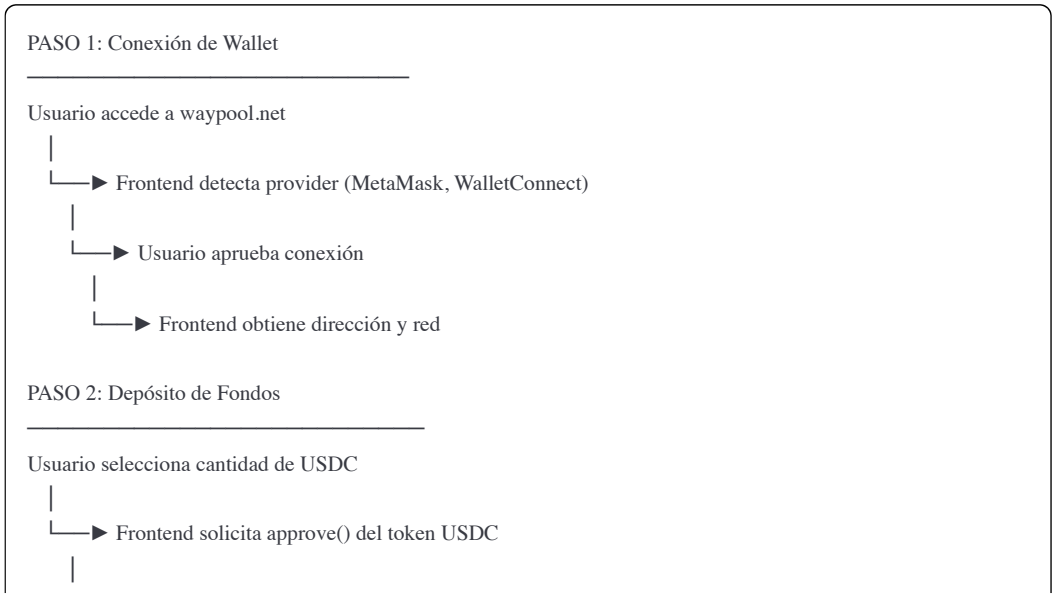
```
            token1: pool.token1(),
            fee: pool.fee(),
            liquidity: pool.liquidity(),
            sqrtPriceX96: sqrtPriceX96,
            tick: tick,
            feeGrowthGlobal0X128: pool.feeGrowthGlobal0X128(),
            feeGrowthGlobal1X128: pool.feeGrowthGlobal1X128()
        });
    }

    /**
     * @notice Obtiene métricas de múltiples pools
     */
    function getMultiplePoolMetrics(
        address[] calldata poolAddresses
    ) external view returns (PoolMetrics[] memory) {
        PoolMetrics[] memory results = new PoolMetrics[](poolAddresses.length);

        for (uint256 i = 0; i < poolAddresses.length; i++) {
            IUniswapV3Pool pool = IUniswapV3Pool(poolAddresses[i]);
            (uint160 sqrtPriceX96, int24 tick,,,,,) = pool.slot0();

            results[i] = PoolMetrics({
                poolAddress: poolAddresses[i],
                token0: pool.token0(),
                token1: pool.token1(),
                fee: pool.fee(),
                liquidity: pool.liquidity(),
                sqrtPriceX96: sqrtPriceX96,
                tick: tick,
                feeGrowthGlobal0X128: pool.feeGrowthGlobal0X128(),
                feeGrowthGlobal1X128: pool.feeGrowthGlobal1X128()
            });
        }

        return results;
    }

    /**
     * @notice Calcula el precio actual de una pool
     */
    function getCurrentPrice(address poolAddress) external view returns (uint256) {
```

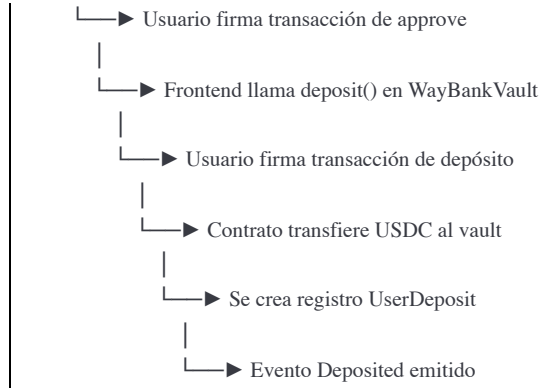
```
        IUniswapV3Pool pool = IUniswapV3Pool(poolAddress);
        (uint160 sqrtPriceX96,,,,,) = pool.slot0();

        // price = (sqrtPriceX96 / 2^96)^2
        uint256 price = uint256(sqrtPriceX96) * uint256(sqrtPriceX96) / (1 << 192);
        return price;
    }

    /**
     * @notice Obtiene la dirección de una pool por sus tokens y fee
     */
    function getPoolAddress(
        address tokenA,
        address tokenB,
        uint24 fee
    ) external view returns (address) {
        return factory.getPool(tokenA, tokenB, fee);
    }
}
```

---

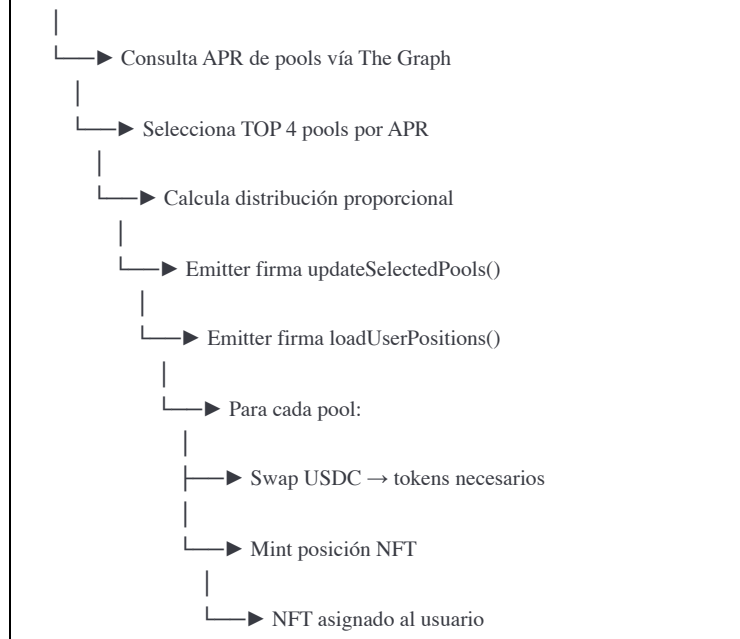
4. Flujo de Usuario

4.1 Paso a Paso Detallado

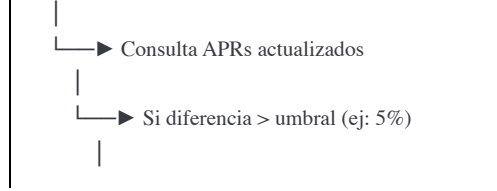




PASO 3: Procesamiento Backend (Emitter Wallet)



PASO 4: Monitoreo y Rebalanceo



```
        └──▶ Emitter firma rebalance()
             │
             └──▶ Ajusta posiciones según nuevos APRs

PASO 5: Retiro
──────────────────────

Usuario solicita retiro
  │
  └──▶ Frontend llama withdraw()
       │
       └──▶ Contrato:
            │
            ├──▶ Recolecta fees de todas las posiciones
            │
            ├──▶ Remueve liquidez
            │
            ├──▶ Quema NFTs
            │
            ├──▶ Convierte todos los tokens a USDC
            │
            └──▶ Transfiere USDC al usuario
```

4.2 Interacción con el NFT

El NFT de posición de Uniswap V3 contiene:

```javascript
```

```
// Datos del NFT de posición
{
  tokenId: 123456,
  nonce: 0,
  operator: "0x...",
  token0: "0xA0b86...USDC",
  token1: "0xC02aa...WETH",
  fee: 3000,
  tickLower: -887220,
  tickUpper: 887220,
  liquidity: "1000000000000000000",
  feeGrowthInside0LastX128: "...",
  feeGrowthInside1LastX128: "...",
  tokensOwed0: 0,
  tokensOwed1: 0
}
```

---

5. Obtención de Datos de Pools (APR)

5.1 The Graph Subgraph Queries

```javascript
```

```
// backend/services/poolAnalyzer.js

const { request, gql } = require('graphql-request');

// Endpoints de subgraphs
const SUBGRAPH_ENDPOINTS = {
  ethereum: 'https://gateway.thegraph.com/api/[API-KEY]/subgraphs/id/5zvR82QoaXYFyDEKLZ9t6v9adgnptxYpKp
  polygon: 'https://gateway.thegraph.com/api/[API-KEY]/subgraphs/id/3hCPRGf4z88VC5rsBKU5AA9FBBq5nF3jbK
  arbitrum: 'https://gateway.thegraph.com/api/[API-KEY]/subgraphs/id/FbCGRftH4a3yZugY7TnbYgPJVEv2LvMT6o
};

/**
 * Obtiene las top pools por volumen y TVL
 */
async function getTopPools(network = 'ethereum', limit = 20) {
  const endpoint = SUBGRAPH_ENDPOINTS[network];

  const query = gql`
    query TopPools($first: Int!) {
      pools(
        first: $first
        orderBy: totalValueLockedUSD
        orderDirection: desc
        where: { totalValueLockedUSD_gt: "100000" }
      ) {
        id
        token0 {
          id
          symbol
          decimals
        }
        token1 {
          id
          symbol
          decimals
        }
        feeTier
        liquidity
        sqrtPrice
        tick
        totalValueLockedUSD
```

```
      totalValueLockedToken0
      totalValueLockedToken1
      volumeUSD
      feesUSD
      txCount
      poolDayData(first: 7, orderBy: date, orderDirection: desc) {
        date
        volumeUSD
        feesUSD
        tvlUSD
      }
    }
  }
`;

  const data = await request(endpoint, query, { first: limit });
  return data.pools;
}

/**
 * Calcula el APR estimado de una pool
 * APR = (fees_24h * 365 / TVL) * 100
 */
function calculatePoolAPR(pool) {
  const dayData = pool.poolDayData;
  if (!dayData || dayData.length === 0) return 0;

  // Promedio de fees diarios de los últimos 7 días
  const avgDailyFees = dayData.reduce((sum, day) =>
    sum + parseFloat(day.feesUSD), 0) / dayData.length;

  const tvl = parseFloat(pool.totalValueLockedUSD);
  if (tvl === 0) return 0;

  // APR anualizado
  const apr = (avgDailyFees * 365 / tvl) * 100;

  return apr;
}

/**
 * Obtiene las 4 mejores pools por APR
```

```
 */
async function getTop4PoolsByAPR(network = 'ethereum') {
  const pools = await getTopPools(network, 50);

  // Calcular APR para cada pool
  const poolsWithAPR = pools.map(pool => ({
    ...pool,
    estimatedAPR: calculatePoolAPR(pool)
  }));

  // Ordenar por APR y tomar las top 4
  const top4 = poolsWithAPR
    .sort((a, b) => b.estimatedAPR - a.estimatedAPR)
    .slice(0, 4);

  return top4;
}

/**
 * Query para datos históricos de una pool específica
 */
async function getPoolHistoricalData(poolId, network = 'ethereum', days = 30) {
  const endpoint = SUBGRAPH_ENDPOINTS[network];

  const query = gql`
    query PoolHistory($poolId: ID!, $days: Int!) {
      poolDayDatas(
        first: $days
        orderBy: date
        orderDirection: desc
        where: { pool: $poolId }
      ) {
        date
        volumeUSD
        feesUSD
        tvlUSD
        open
        high
        low
        close
      }
    }
```

```
  `;

  const data = await request(endpoint, query, { poolId, days });
  return data.poolDayDatas;
}

module.exports = {
  getTopPools,
  calculatePoolAPR,
  getTop4PoolsByAPR,
  getPoolHistoricalData
};
```

5.2 Query de Ejemplo para The Graph

```graphql
```

```
# Query para obtener las pools más rentables
{
  pools(
    first: 10
    orderBy: feesUSD
    orderDirection: desc
    where: {
      totalValueLockedUSD_gt: "1000000"
    }
  ) {
    id
    token0 {
      id
      symbol
      name
      decimals
    }
    token1 {
      id
      symbol
      name
      decimals
    }
    feeTier
    totalValueLockedUSD
    volumeUSD
    feesUSD
    token0Price
    token1Price
  }
}

# Query para datos diarios de una pool
{
  poolDayDatas(
    first: 30
    orderBy: date
    orderDirection: desc
    where: {
      pool: "0x8ad599c3a0ff1de082011efddc58f1908eb6e6d8"
    }
```

```
  ) {
    date
    volumeUSD
    feesUSD
    tvlUSD
    liquidity
  }
}
```

5.3 Cálculo de APR On-Chain vs Off-Chain

javascript

```
/**
 * Método híbrido: combina datos on-chain con históricos
 */
async function getComprehensiveAPR(poolAddress, provider) {
  // 1. Datos on-chain actuales
  const poolContract = new ethers.Contract(poolAddress, POOL_ABI, provider);

  const [
    liquidity,
    slot0,
    feeGrowthGlobal0,
    feeGrowthGlobal1
  ] = await Promise.all([
    poolContract.liquidity(),
    poolContract.slot0(),
    poolContract.feeGrowthGlobal0X128(),
    poolContract.feeGrowthGlobal1X128()
  ]);

  // 2. Datos históricos del subgraph
  const historicalData = await getPoolHistoricalData(poolAddress);

  // 3. Calcular APR basado en fees acumulados
  const dailyFees = historicalData.slice(0, 7).reduce((sum, d) =>
    sum + parseFloat(d.feesUSD), 0) / 7;

  const tvl = parseFloat(historicalData[0]?.tvlUSD || 0);

  const apr = tvl > 0 ? (dailyFees * 365 / tvl) * 100 : 0;

  return {
    apr,
    liquidity: liquidity.toString(),
    currentTick: slot0.tick,
    sqrtPriceX96: slot0.sqrtPriceX96.toString(),
    tvl,
    dailyFees
  };
}
```

6. Sistema de Rebalanceo

6.1 Lógica de Rebalanceo

javascript

```
// backend/services/rebalancer.js

const ethers = require('ethers');

class Rebalancer {
  constructor(config) {
    this.provider = new ethers.providers.JsonRpcProvider(config.rpcUrl);
    this.emitterWallet = new ethers.Wallet(config.emitterPrivateKey, this.provider);
    this.vaultContract = new ethers.Contract(
      config.vaultAddress,
      WAYBANK_ABI,
      this.emitterWallet
    );

    this.rebalanceThreshold = config.threshold || 5; // 5% diferencia mínima
    this.minRebalanceInterval = config.interval || 3600; // 1 hora
  }

  /**
   * Verifica si se necesita rebalanceo
   */
  async needsRebalancing() {
    // Obtener pools actuales del contrato
    const currentPools = await this.vaultContract.getSelectedPools();

    // Obtener APRs actualizados
    const newTop4 = await getTop4PoolsByAPR();

    // Comparar
    let significantChange = false;

    for (let i = 0; i < 4; i++) {
      const currentAPR = currentPools[i].apr.toNumber() / 100;
      const newAPR = newTop4[i]?.estimatedAPR || 0;

      const diff = Math.abs(currentAPR - newAPR);
      const percentDiff = (diff / currentAPR) * 100;

      if (percentDiff > this.rebalanceThreshold) {
        significantChange = true;
        break;
```

```
    }

    // También verificar si cambió el ranking
    if (currentPools[i].poolAddress.toLowerCase() !==
        newTop4[i]?.id.toLowerCase()) {
      significantChange = true;
      break;
    }
  }

  return significantChange;
}

/**
 * Ejecuta el rebalanceo
 */
async executeRebalance() {
  console.log('🔄 Iniciando rebalanceo...');

  // 1. Obtener nuevos APRs
  const newTop4 = await getTop4PoolsByAPR();

  // 2. Preparar arrays para el contrato
  const poolAddresses = newTop4.map(p => p.id);
  const aprs = newTop4.map(p => Math.floor(p.estimatedAPR * 100)); // basis points

  // Rellenar con zeros si hay menos de 4
  while (poolAddresses.length < 4) {
    poolAddresses.push(ethers.constants.AddressZero);
    aprs.push(0);
  }

  // 3. Actualizar pools en el contrato
  const tx1 = await this.vaultContract.updateSelectedPools(poolAddresses, aprs, {
    gasLimit: 500000
  });
  await tx1.wait();
  console.log('✅ Pools actualizadas');

  // 4. Ejecutar rebalanceo
  const tx2 = await this.vaultContract.rebalance({
    gasLimit: 1000000
```

```
    });
    await tx2.wait();
    console.log('✅ Rebalanceo completado');

    return {
      success: true,
      pools: poolAddresses,
      aprs: aprs.map(a => a / 100),
      txHash: tx2.hash
    };
  }

  /**
   * Cron job de rebalanceo
   */
  async runRebalanceJob() {
    try {
      const needs = await this.needsRebalancing();

      if (needs) {
        const result = await this.executeRebalance();
        console.log('📊 Rebalanceo ejecutado:', result);
        return result;
      } else {
        console.log('ℹ️ No se necesita rebalanceo');
        return { success: true, rebalanced: false };
      }
    } catch (error) {
      console.error('❌ Error en rebalanceo:', error);
      throw error;
    }
  }
}

module.exports = Rebalancer;
```

6.2 Cron Job de Rebalanceo

```
javascript
```

```
// backend/jobs/rebalanceCron.js

const cron = require('node-cron');
const Rebalancer = require('../services/rebalancer');

const config = {
  rpcUrl: process.env.RPC_URL,
  emitterPrivateKey: process.env.EMITTER_PRIVATE_KEY,
  vaultAddress: process.env.VAULT_ADDRESS,
  threshold: 5,     // 5% diferencia para rebalancear
  interval: 3600    // Mínimo 1 hora entre rebalanceos
};

const rebalancer = new Rebalancer(config);

// Ejecutar cada hora
cron.schedule('0 * * * *', async () => {
  console.log('⏰ Cron: Verificando rebalanceo...');
  await rebalancer.runRebalanceJob();
});

// También permitir ejecución manual
module.exports = {
  triggerRebalance: () => rebalancer.executeRebalance(),
  checkRebalance: () => rebalancer.needsRebalancing()
};
```

---

7. Gestión de NFTs de Posición

7.1 Mintear NFT con Posición Vacía

En Uniswap V3, no puedes crear un NFT "vacío" directamente. El NFT se crea cuando minteas una posición con liquidez. Sin embargo, podemos crear una posición con liquidez mínima:

```
solidity
```

```
/**
 * @notice Crea una posición con liquidez mínima (casi vacía)
 * @dev El usuario recibe el NFT que luego será cargado por el emitter
 */
function mintMinimalPosition(
    address token0,
    address token1,
    uint24 fee,
    address recipient
) external returns (uint256 tokenId) {
    IUniswapV3Pool pool = IUniswapV3Pool(
        factory.getPool(token0, token1, fee)
    );

    (uint160 sqrtPriceX96, int24 currentTick,,,,) = pool.slot0();
    int24 tickSpacing = pool.tickSpacing();

    // Tick range amplio para posición pasiva
    int24 tickLower = ((currentTick - 10000) / tickSpacing) * tickSpacing;
    int24 tickUpper = ((currentTick + 10000) / tickSpacing) * tickSpacing;

    // Cantidad mínima (1 wei de cada token)
    uint256 amount0Min = 1;
    uint256 amount1Min = 1;

    IERC20(token0).safeTransferFrom(msg.sender, address(this), amount0Min);
    IERC20(token1).safeTransferFrom(msg.sender, address(this), amount1Min);

    IERC20(token0).safeApprove(address(positionManager), amount0Min);
    IERC20(token1).safeApprove(address(positionManager), amount1Min);

    INonfungiblePositionManager.MintParams memory params =
        INonfungiblePositionManager.MintParams({
            token0: token0,
            token1: token1,
            fee: fee,
            tickLower: tickLower,
            tickUpper: tickUpper,
            amount0Desired: amount0Min,
            amount1Desired: amount1Min,
            amount0Min: 0,
```

```
        amount1Min: 0,
        recipient: recipient,
        deadline: block.timestamp + 15 minutes
    });

    (tokenId,,,) = positionManager.mint(params);
}
```

7.2 Incrementar Liquidez del NFT

solidity

```
/**
 * @notice El emitter carga liquidez en el NFT del usuario
 * @param tokenId ID del NFT a cargar
 * @param amount0Desired Cantidad de token0 a añadir
 * @param amount1Desired Cantidad de token1 a añadir
 */
function increaseLiquidity(
    uint256 tokenId,
    uint256 amount0Desired,
    uint256 amount1Desired
) external onlyEmitter returns (uint128 liquidity, uint256 amount0, uint256 amount1) {
    // Obtener info de la posición
    (,, address token0, address token1,,,,,,,,) = positionManager.positions(tokenId);

    // Transferir tokens al contrato
    IERC20(token0).safeTransferFrom(msg.sender, address(this), amount0Desired);
    IERC20(token1).safeTransferFrom(msg.sender, address(this), amount1Desired);

    // Aprobar
    IERC20(token0).safeApprove(address(positionManager), amount0Desired);
    IERC20(token1).safeApprove(address(positionManager), amount1Desired);

    INonfungiblePositionManager.IncreaseLiquidityParams memory params =
        INonfungiblePositionManager.IncreaseLiquidityParams({
            tokenId: tokenId,
            amount0Desired: amount0Desired,
            amount1Desired: amount1Desired,
            amount0Min: 0,
            amount1Min: 0,
            deadline: block.timestamp + 15 minutes
        });

    (liquidity, amount0, amount1) = positionManager.increaseLiquidity(params);
}
```

7.3 Transferir NFT al Usuario

```
javascript
```

```
// En el frontend o backend
async function transferNFTToUser(tokenId, userAddress, signer) {
  const positionManager = new ethers.Contract(
    POSITION_MANAGER_ADDRESS,
    INonfungiblePositionManager.abi,
    signer
  );

  // safeTransferFrom transfiere el NFT
  const tx = await positionManager.safeTransferFrom(
    signer.address,  // from
    userAddress,     // to
    tokenId          // tokenId
  );

  await tx.wait();
  console.log(`NFT ${tokenId} transferido a ${userAddress}`);
}
```

---

8. Swaps Automáticos

8.1 SwapExecutor Contract

solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol";
import "@uniswap/v3-periphery/contracts/interfaces/IQuoter.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

contract SwapExecutor {
    using SafeERC20 for IERC20;

    ISwapRouter public immutable swapRouter;
    IQuoter public immutable quoter;

    // Fee tiers para buscar mejor ruta
    uint24[] public feeTiers = [100, 500, 3000, 10000];

    constructor(address _swapRouter, address _quoter) {
        swapRouter = ISwapRouter(_swapRouter);
        quoter = IQuoter(_quoter);
    }

    /**
     * @notice Obtiene la mejor cotización para un swap
     */
    function getBestQuote(
        address tokenIn,
        address tokenOut,
        uint256 amountIn
    ) external returns (uint256 bestAmountOut, uint24 bestFee) {
        bestAmountOut = 0;
        bestFee = 3000; // default

        for (uint256 i = 0; i < feeTiers.length; i++) {
            try quoter.quoteExactInputSingle(
                tokenIn,
                tokenOut,
                feeTiers[i],
                amountIn,
                0
            ) returns (uint256 amountOut) {
```

```
            if (amountOut > bestAmountOut) {
                bestAmountOut = amountOut;
                bestFee = feeTiers[i];
            }
        } catch {
            // Pool no existe para este fee tier
            continue;
        }
    }
}

/**
 * @notice Ejecuta un swap single-hop optimizado
 */
function executeSwap(
    address tokenIn,
    address tokenOut,
    uint256 amountIn,
    uint256 amountOutMinimum,
    address recipient
) external returns (uint256 amountOut) {
    // Obtener mejor fee tier
    (, uint24 bestFee) = this.getBestQuote(tokenIn, tokenOut, amountIn);

    // Transferir tokens al contrato
    IERC20(tokenIn).safeTransferFrom(msg.sender, address(this), amountIn);
    IERC20(tokenIn).safeApprove(address(swapRouter), amountIn);

    ISwapRouter.ExactInputSingleParams memory params =
        ISwapRouter.ExactInputSingleParams({
            tokenIn: tokenIn,
            tokenOut: tokenOut,
            fee: bestFee,
            recipient: recipient,
            deadline: block.timestamp + 15 minutes,
            amountIn: amountIn,
            amountOutMinimum: amountOutMinimum,
            sqrtPriceLimitX96: 0
        });

    amountOut = swapRouter.exactInputSingle(params);
}
```

```
/**
 * @notice Ejecuta un swap multi-hop
 * @param path Encoded path (token, fee, token, fee, token, ...)
 */
function executeMultiHopSwap(
    bytes memory path,
    uint256 amountIn,
    uint256 amountOutMinimum,
    address recipient
) external returns (uint256 amountOut) {
    // Decodificar primer token del path
    address tokenIn;
    assembly {
        tokenIn := mload(add(path, 20))
    }

    IERC20(tokenIn).safeTransferFrom(msg.sender, address(this), amountIn);
    IERC20(tokenIn).safeApprove(address(swapRouter), amountIn);

    ISwapRouter.ExactInputParams memory params =
        ISwapRouter.ExactInputParams({
            path: path,
            recipient: recipient,
            deadline: block.timestamp + 15 minutes,
            amountIn: amountIn,
            amountOutMinimum: amountOutMinimum
        });

    amountOut = swapRouter.exactInput(params);
}

/**
 * @notice Prepara tokens para una pool específica
 * @dev Convierte USDC a los dos tokens necesarios para la pool
 */
function prepareTokensForPool(
    address usdcAddress,
    address token0,
    address token1,
    uint256 usdcAmount
) external returns (uint256 amount0, uint256 amount1) {
```

```
        // Dividir USDC 50/50 (simplificado, en producción usar cálculo preciso)
        uint256 halfAmount = usdcAmount / 2;

        // Si token0 no es USDC, swap
        if (token0 != usdcAddress) {
            amount0 = this.executeSwap(
                usdcAddress,
                token0,
                halfAmount,
                0, // Sin mínimo por simplicidad
                address(this)
            );
        } else {
            amount0 = halfAmount;
        }

        // Si token1 no es USDC, swap
        if (token1 != usdcAddress) {
            amount1 = this.executeSwap(
                usdcAddress,
                token1,
                halfAmount,
                0,
                address(this)
            );
        } else {
            amount1 = halfAmount;
        }

        // Transferir tokens al caller
        IERC20(token0).safeTransfer(msg.sender, amount0);
        IERC20(token1).safeTransfer(msg.sender, amount1);
    }
}
```

8.2 Encoding del Path para Multi-hop

```
javascript
```

```javascript
// utils/pathEncoder.js

const ethers = require('ethers');

/**
 * Codifica un path para swaps multi-hop
 * @param tokens Array de direcciones de tokens
 * @param fees Array de fee tiers (debe tener longitud tokens.length - 1)
 * @returns Encoded path
 */
function encodePath(tokens, fees) {
  if (tokens.length - 1 !== fees.length) {
    throw new Error('tokens/fees length mismatch');
  }

  let encoded = '0x';

  for (let i = 0; i < fees.length; i++) {
    // 20 bytes de dirección
    encoded += tokens[i].slice(2).toLowerCase();
    // 3 bytes de fee
    encoded += fees[i].toString(16).padStart(6, '0');
  }

  // Último token
  encoded += tokens[tokens.length - 1].slice(2).toLowerCase();

  return encoded;
}

/**
 * Encuentra la mejor ruta de swap
 */
async function findBestRoute(tokenIn, tokenOut, amountIn, provider) {
  const quoterContract = new ethers.Contract(
    QUOTER_V2_ADDRESS,
    IQuoterV2.abi,
    provider
  );

  const routes = [
```

```
  // Ruta directa
  { path: [tokenIn, tokenOut], fees: [3000] },
  { path: [tokenIn, tokenOut], fees: [500] },
  { path: [tokenIn, tokenOut], fees: [10000] },

  // Rutas via WETH
  { path: [tokenIn, WETH, tokenOut], fees: [3000, 3000] },
  { path: [tokenIn, WETH, tokenOut], fees: [500, 500] },

  // Rutas via USDC
  { path: [tokenIn, USDC, tokenOut], fees: [500, 500] },
];

let bestRoute = null;
let bestAmountOut = ethers.BigNumber.from(0);

for (const route of routes) {
  try {
    const encoded = encodePath(route.path, route.fees);
    const amountOut = await quoterContract.callStatic.quoteExactInput(
      encoded,
      amountIn
    );

    if (amountOut.gt(bestAmountOut)) {
      bestAmountOut = amountOut;
      bestRoute = { ...route, encodedPath: encoded, amountOut };
    }
  } catch (e) {
    // Ruta no disponible
    continue;
  }
 }

 return bestRoute;
}

module.exports = { encodePath, findBestRoute };
```

9. Backend y API

9.1 Estructura del Backend

```
backend/
├── src/
│   ├── index.js              # Entry point
│   ├── config/
│   │   ├── index.js          # Configuración general
│   │   └── contracts.js      # ABIs y direcciones
│   ├── services/
│   │   ├── poolAnalyzer.js   # Análisis de pools
│   │   ├── rebalancer.js     # Lógica de rebalanceo
│   │   ├── positionManager.js  # Gestión de posiciones
│   │   └── blockchain.js     # Interacción con blockchain
│   ├── jobs/
│   │   ├── rebalanceCron.js  # Cron de rebalanceo
│   │   └── depositWatcher.js  # Watcher de depósitos
│   ├── routes/
│   │   ├── pools.js          # Endpoints de pools
│   │   ├── users.js          # Endpoints de usuarios
│   │   └── admin.js          # Endpoints admin
│   └── middleware/
│       ├── auth.js           # Autenticación
│       └── validation.js     # Validación
├── .env
├── package.json
└── Dockerfile
```

9.2 API Endpoints

```javascript
```

```
// backend/src/routes/pools.js

const express = require('express');
const router = express.Router();
const poolAnalyzer = require('../services/poolAnalyzer');

/**
 * GET /api/pools/top
 * Obtiene las top 4 pools por APR
 */
router.get('/top', async (req, res) => {
  try {
    const network = req.query.network || 'ethereum';
    const pools = await poolAnalyzer.getTop4PoolsByAPR(network);

    res.json({
      success: true,
      data: pools.map(pool => ({
        address: pool.id,
        token0: pool.token0,
        token1: pool.token1,
        feeTier: pool.feeTier,
        tvl: pool.totalValueLockedUSD,
        volume24h: pool.poolDayData[0]?.volumeUSD || 0,
        fees24h: pool.poolDayData[0]?.feesUSD || 0,
        apr: pool.estimatedAPR
      }))
    });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});

/**
 * GET /api/pools/:address
 * Obtiene detalles de una pool específica
 */
router.get('/:address', async (req, res) => {
  try {
    const { address } = req.params;
    const network = req.query.network || 'ethereum';
```

```
    const poolData = await poolAnalyzer.getPoolDetails(address, network);
    const historicalData = await poolAnalyzer.getPoolHistoricalData(address, network);

    res.json({
      success: true,
      data: {
        ...poolData,
        historical: historicalData
      }
    });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});

/**
 * GET /api/pools/distribution/:amount
 * Calcula la distribución óptima para un monto dado
 */
router.get('/distribution/:amount', async (req, res) => {
  try {
    const amount = parseFloat(req.params.amount);
    const network = req.query.network || 'ethereum';

    const pools = await poolAnalyzer.getTop4PoolsByAPR(network);

    // Calcular distribución proporcional al APR
    const totalAPR = pools.reduce((sum, p) => sum + p.estimatedAPR, 0);

    const distribution = pools.map(pool => ({
      pool: pool.id,
      apr: pool.estimatedAPR,
      percentage: (pool.estimatedAPR / totalAPR) * 100,
      allocation: (pool.estimatedAPR / totalAPR) * amount
    }));

    res.json({
      success: true,
      data: {
        totalAmount: amount,
        pools: distribution,
```

```
      estimatedWeightedAPR: totalAPR / pools.length
    }
  });
 } catch (error) {
  res.status(500).json({ success: false, error: error.message });
 }
});

module.exports = router;
```

9.3 Watcher de Eventos

```javascript
```

```
// backend/src/jobs/depositWatcher.js

const ethers = require('ethers');
const { WAYBANK_ABI, VAULT_ADDRESS } = require('../config/contracts');

class DepositWatcher {
  constructor(provider, emitterWallet, poolAnalyzer) {
    this.provider = provider;
    this.emitterWallet = emitterWallet;
    this.poolAnalyzer = poolAnalyzer;

    this.vaultContract = new ethers.Contract(
      VAULT_ADDRESS,
      WAYBANK_ABI,
      emitterWallet
    );
  }

  /**
   * Inicia el watcher de eventos Deposited
   */
  start() {
    console.log('👀 Iniciando watcher de depósitos...');

    this.vaultContract.on('Deposited', async (user, amount, timestamp, event) => {
      console.log(`💰 Nuevo depósito detectado:`);
      console.log(`   Usuario: ${user}`);
      console.log(`   Cantidad: ${ethers.utils.formatUnits(amount, 6)} USDC`);
      console.log(`   Timestamp: ${new Date(timestamp * 1000).toISOString()}`);

      await this.processDeposit(user, amount);
    });
  }

  /**
   * Procesa un nuevo depósito
   */
  async processDeposit(user, amount) {
    try {
      // 1. Obtener las mejores pools
      const top4 = await this.poolAnalyzer.getTop4PoolsByAPR();
```

```
    // 2. Calcular distribución
    const totalAPR = top4.reduce((sum, p) => sum + p.estimatedAPR, 0);
    const allocations = top4.map(pool => ({
      pool: pool.id,
      amount: amount.mul(Math.floor(pool.estimatedAPR * 100)).div(Math.floor(totalAPR * 100))
    }));

    // 3. Actualizar pools si es necesario
    const poolAddresses = top4.map(p => p.id);
    const aprs = top4.map(p => Math.floor(p.estimatedAPR * 100));

    // Rellenar arrays
    while (poolAddresses.length < 4) {
      poolAddresses.push(ethers.constants.AddressZero);
      aprs.push(0);
    }

    const tx1 = await this.vaultContract.updateSelectedPools(poolAddresses, aprs);
    await tx1.wait();

    // 4. Cargar posiciones del usuario
    const tx2 = await this.vaultContract.loadUserPositions(
      user,
      allocations.map(a => a.pool),
      allocations.map(a => a.amount)
    );
    await tx2.wait();

    console.log(`✅ Posiciones creadas para ${user}`);

  } catch (error) {
    console.error(`❌ Error procesando depósito de ${user}:`, error);
    // Implementar retry o notificación
  }
 }

 stop() {
  this.vaultContract.removeAllListeners('Deposited');
  console.log('🛑 Watcher detenido');
 }
}
```

```
module.exports = DepositWatcher;
```

---

10. Frontend (waypool.net)

10.1 Estructura del Frontend

```
frontend/
├── src/
│   ├── App.jsx
│   ├── index.js
│   ├── components/
│   │   ├── WalletConnect.jsx      # Conexión de wallet
│   │   ├── DepositForm.jsx        # Formulario de depósito
│   │   ├── PoolsDisplay.jsx       # Muestra pools seleccionadas
│   │   ├── PositionCard.jsx       # Muestra posición del usuario
│   │   └── Dashboard.jsx          # Dashboard principal
│   ├── hooks/
│   │   ├── useWallet.js           # Hook para wallet
│   │   ├── useContract.js         # Hook para contratos
│   │   └── usePools.js            # Hook para datos de pools
│   ├── services/
│   │   ├── api.js                 # Cliente API
│   │   └── web3.js                # Utilidades Web3
│   ├── config/
│   │   └── contracts.js           # Direcciones y ABIs
│   └── styles/
│       └── main.css
├── public/
├── package.json
└── vite.config.js
```

10.2 Componente Principal

```jsx
```

```jsx
// frontend/src/components/Dashboard.jsx

import React, { useState, useEffect } from 'react';
import { ethers } from 'ethers';
import { useWallet } from '../hooks/useWallet';
import { useContract } from '../hooks/useContract';
import WalletConnect from './WalletConnect';
import DepositForm from './DepositForm';
import PoolsDisplay from './PoolsDisplay';
import PositionCard from './PositionCard';
import { getTopPools } from '../services/api';

const Dashboard = () => {
  const { address, isConnected, connect, disconnect } = useWallet();
  const { vaultContract, usdcContract } = useContract();

  const [pools, setPools] = useState([]);
  const [userDeposit, setUserDeposit] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  // Cargar pools al montar
  useEffect(() => {
    const fetchPools = async () => {
      try {
        const data = await getTopPools();
        setPools(data);
      } catch (err) {
        setError('Error cargando pools');
      }
    };
    fetchPools();
  }, []);

  // Cargar depósito del usuario cuando conecta
  useEffect(() => {
    const fetchUserDeposit = async () => {
      if (!address || !vaultContract) return;

      try {
        const deposit = await vaultContract.getUserDeposit(address);
```

```
      if (deposit.isActive) {
        setUserDeposit({
          amount: ethers.utils.formatUnits(deposit.amount, 6),
          depositTime: new Date(deposit.depositTime.toNumber() * 1000),
          positionIds: deposit.positionIds.map(id => id.toString())
        });
      }
    } catch (err) {
      console.error('Error fetching deposit:', err);
    }
  };

  fetchUserDeposit();
}, [address, vaultContract]);

// Handler de depósito
const handleDeposit = async (amount) => {
  if (!vaultContract || !usdcContract) return;

  setLoading(true);
  setError(null);

  try {
    const amountWei = ethers.utils.parseUnits(amount.toString(), 6);

    // 1. Aprobar USDC
    const approveTx = await usdcContract.approve(
      vaultContract.address,
      amountWei
    );
    await approveTx.wait();

    // 2. Depositar
    const depositTx = await vaultContract.deposit(amountWei);
    await depositTx.wait();

    // 3. Actualizar UI
    setUserDeposit({
      amount: amount.toString(),
      depositTime: new Date(),
      positionIds: []
    });
```

```
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
};

// Handler de retiro
const handleWithdraw = async () => {
  if (!vaultContract) return;

  setLoading(true);
  setError(null);

  try {
    const tx = await vaultContract.withdraw();
    await tx.wait();
    setUserDeposit(null);
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
};

return (
  <div className="dashboard">
    <header>
      <h1>WayPool - Liquidez Inteligente</h1>
      <WalletConnect
        address={address}
        isConnected={isConnected}
        onConnect={connect}
        onDisconnect={disconnect}
      />
    </header>

    <main>
      {error && <div className="error">{error}</div>}

      {/* Pools Actuales */}
```

```
      <section className="pools-section">
        <h2>Top 4 Pools por APR</h2>
        <PoolsDisplay pools={pools} />
      </section>

      {/* Área de Usuario */}
      {isConnected && (
        <section className="user-section">
          {!userDeposit ? (
            <DepositForm
              onDeposit={handleDeposit}
              loading={loading}
              pools={pools}
            />
          ) : (
            <div className="position-section">
              <h2>Tu Posición</h2>
              <PositionCard
                deposit={userDeposit}
                onWithdraw={handleWithdraw}
                loading={loading}
              />
            </div>
          )}
        </section>
      )}

      {/* Calculadora de Distribución */}
      <section className="calculator-section">
        <h2>Calculadora de Distribución</h2>
        <DistributionCalculator pools={pools} />
      </section>
    </main>
  </div>
 );
};

export default Dashboard;
```

10.3 Hook de Wallet

javascript

```
// frontend/src/hooks/useWallet.js

import { useState, useEffect, useCallback } from 'react';
import { ethers } from 'ethers';

export const useWallet = () => {
  const [address, setAddress] = useState(null);
  const [provider, setProvider] = useState(null);
  const [signer, setSigner] = useState(null);
  const [chainId, setChainId] = useState(null);
  const [isConnected, setIsConnected] = useState(false);

  // Verificar conexión existente al cargar
  useEffect(() => {
    const checkConnection = async () => {
      if (typeof window.ethereum !== 'undefined') {
        const web3Provider = new ethers.providers.Web3Provider(window.ethereum);
        const accounts = await web3Provider.listAccounts();

        if (accounts.length > 0) {
          setProvider(web3Provider);
          setSigner(web3Provider.getSigner());
          setAddress(accounts[0]);
          setChainId(await web3Provider.getNetwork().then(n => n.chainId));
          setIsConnected(true);
        }
      }
    };

    checkConnection();
  }, []);

  // Escuchar cambios de cuenta/red
  useEffect(() => {
    if (typeof window.ethereum !== 'undefined') {
      window.ethereum.on('accountsChanged', (accounts) => {
        if (accounts.length > 0) {
          setAddress(accounts[0]);
        } else {
          disconnect();
        }
```

```
    });

    window.ethereum.on('chainChanged', (newChainId) => {
      setChainId(parseInt(newChainId, 16));
      window.location.reload();
    });
  }

  return () => {
    if (typeof window.ethereum !== 'undefined') {
      window.ethereum.removeAllListeners('accountsChanged');
      window.ethereum.removeAllListeners('chainChanged');
    }
  };
}, []);

const connect = useCallback(async () => {
  if (typeof window.ethereum === 'undefined') {
    throw new Error('MetaMask no instalado');
  }

  try {
    const web3Provider = new ethers.providers.Web3Provider(window.ethereum);
    await web3Provider.send('eth_requestAccounts', []);

    const web3Signer = web3Provider.getSigner();
    const userAddress = await web3Signer.getAddress();
    const network = await web3Provider.getNetwork();

    setProvider(web3Provider);
    setSigner(web3Signer);
    setAddress(userAddress);
    setChainId(network.chainId);
    setIsConnected(true);

    return userAddress;
  } catch (error) {
    console.error('Error conectando wallet:', error);
    throw error;
  }
}, []);
```

```
  const disconnect = useCallback(() => {
    setProvider(null);
    setSigner(null);
    setAddress(null);
    setChainId(null);
    setIsConnected(false);
  }, []);

  const switchNetwork = useCallback(async (targetChainId) => {
    if (!window.ethereum) return;

    try {
      await window.ethereum.request({
        method: 'wallet_switchEthereumChain',
        params: [{ chainId: `0x${targetChainId.toString(16)}` }],
      });
    } catch (error) {
      // Si la red no existe, intentar añadirla
      if (error.code === 4902) {
        // Implementar addNetwork si es necesario
      }
      throw error;
    }
  }, []);

  return {
    address,
    provider,
    signer,
    chainId,
    isConnected,
    connect,
    disconnect,
    switchNetwork
  };
};
```

---

11. Seguridad y Mejores Prácticas

11.1 Consideraciones de Seguridad

solidity

```
// Checklist de seguridad para el contrato

// ✅ 1. ReentrancyGuard en todas las funciones que mueven fondos
function deposit(uint256 amount) external nonReentrant { ... }
function withdraw() external nonReentrant { ... }

// ✅ 2. Checks-Effects-Interactions pattern
function withdraw() external nonReentrant {
    // CHECKS
    require(userDeposits[msg.sender].isActive, "No deposit");

    // EFFECTS
    uint256 amount = userDeposits[msg.sender].amount;
    delete userDeposits[msg.sender];
    totalDeposited -= amount;

    // INTERACTIONS
    depositToken.safeTransfer(msg.sender, amount);
}

// ✅ 3. Control de acceso estricto
modifier onlyEmitter() {
    require(msg.sender == emitterWallet, "Not emitter");
    _;
}

// ✅ 4. Validación de inputs
function setEmitterWallet(address _emitter) external onlyOwner {
    require(_emitter != address(0), "Invalid address");
    require(_emitter != emitterWallet, "Same address");
    emitterWallet = _emitter;
}

// ✅ 5. Slippage protection
INonfungiblePositionManager.MintParams memory params = ...{
    amount0Min: (amount0Desired * 95) / 100, // 5% slippage
    amount1Min: (amount1Desired * 95) / 100,
    deadline: block.timestamp + 15 minutes
};

// ✅ 6. Emergency functions
```

```
function emergencyWithdraw(address token) external onlyOwner {
    uint256 balance = IERC20(token).balanceOf(address(this));
    IERC20(token).safeTransfer(owner(), balance);
}

// ✅ 7. Pausable si es necesario
function pause() external onlyOwner {
    _pause();
}
```

11.2 Gestión Segura de Private Keys

javascript

```
// backend/config/security.js

// ❌ NUNCA hacer esto
const privateKey = "0x123...abc"; // Hardcodeado

// ✅ Usar variables de entorno
const privateKey = process.env.EMITTER_PRIVATE_KEY;

// ✅ Mejor aún: usar AWS Secrets Manager o similar
const { SecretsManagerClient, GetSecretValueCommand } = require("@aws-sdk/client-secrets-manager");

async function getEmitterWallet() {
  const client = new SecretsManagerClient({ region: "us-east-1" });

  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: "waybank/emitter-wallet",
    })
  );

  const secret = JSON.parse(response.SecretString);
  return new ethers.Wallet(secret.privateKey);
}

// ✅ También considerar: Hardware wallets (Ledger), Multi-sig (Gnosis Safe)
```

11.3 Monitoreo y Alertas

javascript

```
// backend/services/monitoring.js

const { WebClient } = require('@slack/web-api');

class Monitor {
  constructor() {
    this.slack = new WebClient(process.env.SLACK_TOKEN);
    this.alertChannel = '#waybank-alerts';
  }

  async sendAlert(level, message, details = {}) {
    const emoji = {
      info: 'ℹ️',
      warning: '⚠️',
      error: '🚨',
      success: '✅'
    };

    await this.slack.chat.postMessage({
      channel: this.alertChannel,
      text: `${emoji[level]} *WayBank Alert*\n${message}`,
      attachments: [{
        color: level === 'error' ? 'danger' : level === 'warning' ? 'warning' : 'good',
        fields: Object.entries(details).map(([key, value]) => ({
          title: key,
          value: String(value),
          short: true
        }))
      }]
    });
  }

  // Monitorear transacciones fallidas
  async watchTransactions(vaultContract) {
    vaultContract.on('error', async (error) => {
      await this.sendAlert('error', 'Transaction failed', {
        error: error.message,
        code: error.code
      });
    });
  }
```

```
  // Monitorear balance del emitter
  async checkEmitterBalance(emitterAddress, provider) {
    const balance = await provider.getBalance(emitterAddress);
    const ethBalance = ethers.utils.formatEther(balance);

    if (parseFloat(ethBalance) < 0.1) {
      await this.sendAlert('warning', 'Low emitter ETH balance', {
        address: emitterAddress,
        balance: `${ethBalance} ETH`
      });
    }
  }

  // Monitorear TVL del vault
  async checkVaultTVL(vaultContract) {
    const tvl = await vaultContract.totalDeposited();
    const tvlFormatted = ethers.utils.formatUnits(tvl, 6);

    // Alertar si TVL cae más de 10% en 1 hora
    // (Implementar lógica de comparación con valor anterior)
  }
}

module.exports = Monitor;
```

---

12. Configuración y Despliegue

12.1 Hardhat Configuration

```
javascript
```

```
// hardhat.config.js

require("@nomicfoundation/hardhat-toolbox");
require("@openzeppelin/hardhat-upgrades");
require("dotenv").config();

module.exports = {
 solidity: {
  version: "0.8.19",
  settings: {
   optimizer: {
    enabled: true,
    runs: 200
   },
   viaIR: true
  }
 },
 networks: {
  hardhat: {
   forking: {
    url: process.env.MAINNET_RPC_URL,
    blockNumber: 18500000 // Fijar bloque para tests consistentes
   }
  },
  mainnet: {
   url: process.env.MAINNET_RPC_URL,
   accounts: [process.env.DEPLOYER_PRIVATE_KEY],
   gasPrice: "auto"
  },
  polygon: {
   url: process.env.POLYGON_RPC_URL,
   accounts: [process.env.DEPLOYER_PRIVATE_KEY],
   gasPrice: "auto"
  },
  arbitrum: {
   url: process.env.ARBITRUM_RPC_URL,
   accounts: [process.env.DEPLOYER_PRIVATE_KEY]
  },
  sepolia: {
   url: process.env.SEPOLIA_RPC_URL,
   accounts: [process.env.DEPLOYER_PRIVATE_KEY]
```

```
  }
 },
 etherscan: {
  apiKey: {
   mainnet: process.env.ETHERSCAN_API_KEY,
   polygon: process.env.POLYGONSCAN_API_KEY,
   arbitrumOne: process.env.ARBISCAN_API_KEY
  }
 }
};
```

12.2 Script de Despliegue

javascript

```
// scripts/deploy.js

const { ethers, upgrades } = require("hardhat");

async function main() {
  console.log("🚀 Deploying WayBank v6.0...\n");

  const [deployer] = await ethers.getSigners();
  console.log("Deployer:", deployer.address);
  console.log("Balance:", ethers.utils.formatEther(await deployer.getBalance()), "ETH\n");

  // Direcciones de Uniswap V3 (Mainnet)
  const ADDRESSES = {
    POSITION_MANAGER: "0xC36442b4a4522E871399CD717aBDD847Ab11FE88",
    SWAP_ROUTER: "0xE592427A0AEce92De3Edee1F18E0157C05861564",
    FACTORY: "0x1F98431c8aD98523631AE4a59f267346ea31F984",
    USDC: "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48"
  };

  // Emitter wallet (configurar en .env)
  const EMITTER_WALLET = process.env.EMITTER_WALLET_ADDRESS;

  // 1. Deploy PoolAnalyzer
  console.log("1. Deploying PoolAnalyzer...");
  const PoolAnalyzer = await ethers.getContractFactory("PoolAnalyzer");
  const poolAnalyzer = await PoolAnalyzer.deploy(ADDRESSES.FACTORY);
  await poolAnalyzer.deployed();
  console.log("   PoolAnalyzer deployed to:", poolAnalyzer.address);

  // 2. Deploy SwapExecutor
  console.log("2. Deploying SwapExecutor...");
  const SwapExecutor = await ethers.getContractFactory("SwapExecutor");
  const swapExecutor = await SwapExecutor.deploy(
    ADDRESSES.SWAP_ROUTER,
    "0xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6" // Quoter
  );
  await swapExecutor.deployed();
  console.log("   SwapExecutor deployed to:", swapExecutor.address);

  // 3. Deploy WayBankVault
  console.log("3. Deploying WayBankVault...");
```

```
  const WayBankVault = await ethers.getContractFactory("WayBankVault");
  const vault = await WayBankVault.deploy(
    ADDRESSES.POSITION_MANAGER,
    ADDRESSES.SWAP_ROUTER,
    ADDRESSES.FACTORY,
    ADDRESSES.USDC,
    EMITTER_WALLET
  );
  await vault.deployed();
  console.log("   WayBankVault deployed to:", vault.address);

  // Guardar direcciones
  const deploymentInfo = {
    network: (await ethers.provider.getNetwork()).name,
    deployer: deployer.address,
    contracts: {
      PoolAnalyzer: poolAnalyzer.address,
      SwapExecutor: swapExecutor.address,
      WayBankVault: vault.address
    },
    timestamp: new Date().toISOString()
  };

  console.log("\n✅ Deployment complete!");
  console.log(JSON.stringify(deploymentInfo, null, 2));

  // Guardar a archivo
  const fs = require('fs');
  fs.writeFileSync(
    `deployments/${deploymentInfo.network}.json`,
    JSON.stringify(deploymentInfo, null, 2)
  );
}

main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });
```

12.3 Variables de Entorno (.env)

```bash
# .env - NO COMMITEAR ESTE ARCHIVO

# RPCs
MAINNET_RPC_URL=https://eth-mainnet.g.alchemy.com/v2/YOUR_KEY
POLYGON_RPC_URL=https://polygon-mainnet.g.alchemy.com/v2/YOUR_KEY
ARBITRUM_RPC_URL=https://arb-mainnet.g.alchemy.com/v2/YOUR_KEY
SEPOLIA_RPC_URL=https://eth-sepolia.g.alchemy.com/v2/YOUR_KEY

# Wallets
DEPLOYER_PRIVATE_KEY=0x...
EMITTER_PRIVATE_KEY=0x...
EMITTER_WALLET_ADDRESS=0x...

# APIs
ETHERSCAN_API_KEY=...
POLYGONSCAN_API_KEY=...
ARBISCAN_API_KEY=...
THE_GRAPH_API_KEY=...

# Backend
PORT=3001
DATABASE_URL=mongodb://localhost:27017/waybank

# Monitoring
SLACK_TOKEN=xoxb-...

# Security
JWT_SECRET=...
```

---

13. Código Completo de Contratos

13.1 Interfaces Necesarias

```solidity
```

```solidity
// contracts/interfaces/IWayBankVault.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

interface IWayBankVault {
    struct UserDeposit {
        uint256 amount;
        uint256 depositTime;
        uint256[] positionIds;
        bool isActive;
    }

    struct PoolInfo {
        address poolAddress;
        address token0;
        address token1;
        uint24 fee;
        uint256 apr;
        uint256 allocatedAmount;
        int24 tickLower;
        int24 tickUpper;
    }

    event Deposited(address indexed user, uint256 amount, uint256 timestamp);
    event Withdrawn(address indexed user, uint256 amount, uint256 timestamp);
    event NFTMinted(address indexed user, uint256 tokenId, address pool);
    event PoolsUpdated(address[4] pools, uint256[4] aprs);
    event Rebalanced(uint256 timestamp, uint256[4] newAllocations);

    function deposit(uint256 amount) external returns (uint256 tokenId);
    function withdraw() external;
    function getUserDeposit(address user) external view returns (
        uint256 amount,
        uint256 depositTime,
        uint256[] memory positionIds,
        bool isActive
    );
    function getSelectedPools() external view returns (PoolInfo[4] memory);
    function calculateDistribution(uint256 amount) external view returns (uint256[4] memory);
}
```

📝 Checklist de Implementación

Fase 1: Preparación

- [ ] Configurar entorno de desarrollo (Hardhat, Node.js)
- [ ] Obtener API keys (Alchemy/Infura, The Graph, Etherscan)
- [ ] Crear wallets de despliegue y emitter
- [ ] Fondear wallets con ETH para gas

Fase 2: Smart Contracts

- [ ] Implementar WayBankVault.sol
- [ ] Implementar PoolAnalyzer.sol
- [ ] Implementar SwapExecutor.sol
- [ ] Escribir tests unitarios
- [ ] Desplegar en testnet (Sepolia)
- [ ] Verificar contratos en Etherscan

Fase 3: Backend

- [ ] Configurar servidor Node.js/Express
- [ ] Implementar servicio de análisis de pools
- [ ] Implementar watcher de eventos
- [ ] Implementar cron de rebalanceo
- [ ] Configurar base de datos para logs
- [ ] Implementar sistema de alertas

Fase 4: Frontend

- [ ] Configurar proyecto React/Vite
- [ ] Implementar conexión de wallet
- [ ] Implementar formulario de depósito
- [ ] Implementar visualización de pools
- [ ] Implementar dashboard de usuario
- [ ] Testing en testnet

Fase 5: Producción

- [ ] Auditoría de seguridad (opcional pero recomendado)
- [ ] Despliegue en mainnet

- [ ] Verificación de contratos
- [ ] Configuración de monitoreo
- [ ] Documentación para usuarios
- [ ] Launch en waypool.net

🔗 Recursos Adicionales

- Uniswap V3 Docs
- The Graph Docs
- OpenZeppelin Contracts
- Hardhat Docs
- Ethers.js Docs

Documento preparado para Claude Code
Última actualización: Enero 2026