import express from 'express';

import { ethers } from 'ethers';

import axios from 'axios';

import { config } from 'dotenv';

import { Pool } from 'pg';

import cron from 'node-cron';

config();

const app = express();

app.use(express.json());

app.use(express.static('public'));

const pool = new Pool({

user: 'postgres',

host: 'localhost',

database: 'flash\_loan\_db',

password: 'yourpassword',

port: 5432,

});

const providers: { [key: string]: ethers.providers.JsonRpcProvider } = {

ethereum: new ethers.providers.JsonRpcProvider(process.env.ETHEREUM\_RPC\_URL),

arbitrum: new ethers.providers.JsonRpcProvider(process.env.ARBITRUM\_RPC\_URL),

optimism: new ethers.providers.JsonRpcProvider(process.env.OPTIMISM\_RPC\_URL),

base: new ethers.providers.JsonRpcProvider(process.env.BASE\_RPC\_URL),

polygon: new ethers.providers.JsonRpcProvider(process.env.POLYGON\_RPC\_URL),

rootstock: new ethers.providers.JsonRpcProvider(process.env.ROOTSTOCK\_RPC\_URL),

};

const executorWallet = new ethers.Wallet(process.env.EXECUTOR\_PRIVATE\_KEY!);

const profitWallet = new ethers.Wallet(ethers.utils.randomBytes(32)).address; // Random for now

// Initialize DB

async function initDb() {

await pool.query(`

CREATE TABLE IF NOT EXISTS chains (

id SERIAL PRIMARY KEY,

name VARCHAR(50),

type VARCHAR(20),

chain\_id INT,

symbol VARCHAR(10),

rpc JSONB,

explorer\_url VARCHAR(100)

);

CREATE TABLE IF NOT EXISTS markets (

id SERIAL PRIMARY KEY,

chain\_id INT REFERENCES chains(id),

protocol VARCHAR(20),

address VARCHAR(42),

metadata JSONB

);

CREATE TABLE IF NOT EXISTS pools (

id SERIAL PRIMARY KEY,

market\_id INT REFERENCES markets(id),

token0 VARCHAR(42),

token1 VARCHAR(42),

fee\_bps INT,

liquidity NUMERIC,

last\_updated TIMESTAMP

);

CREATE TABLE IF NOT EXISTS simulations (

id SERIAL PRIMARY KEY,

executor\_contract VARCHAR(42),

chain\_id INT REFERENCES chains(id),

payload\_hex TEXT,

expected\_profit\_wei NUMERIC,

gas\_wei NUMERIC,

flash\_fee\_wei NUMERIC,

result VARCHAR(20),

details JSONB,

ts TIMESTAMP

);

`);

// Insert chain configs

await pool.query(`

INSERT INTO chains (name, type, chain\_id, symbol, rpc, explorer\_url) VALUES

('Ethereum', 'evm', 1, 'ETH', '{"url": "${process.env.ETHEREUM\_RPC\_URL}"}', 'https://etherscan.io'),

('Arbitrum', 'evm', 42161, 'ETH', '{"url": "${process.env.ARBITRUM\_RPC\_URL}"}', 'https://arbiscan.io'),

('Optimism', 'evm', 10, 'ETH', '{"url": "${process.env.OPTIMISM\_RPC\_URL}"}', 'https://optimistic.etherscan.io'),

('Base', 'evm', 8453, 'ETH', '{"url": "${process.env.BASE\_RPC\_URL}"}', 'https://basescan.org'),

('Polygon', 'evm', 137, 'MATIC', '{"url": "${process.env.POLYGON\_RPC\_URL}"}', 'https://polygonscan.com'),

('Rootstock', 'evm', 30, 'RBTC', '{"url": "${process.env.ROOTSTOCK\_RPC\_URL}"}', 'https://explorer.rsk.co')

ON CONFLICT DO NOTHING;

`);

}

// Index flash loan opportunities

async function indexOpportunities() {

const protocols = [

{ name: 'aave', address: '0x87870Bca3F3fD6335C3F4ce8392D69350B4fA4E2', chain: 'ethereum' },

{ name: 'uniswap\_v3', address: '0x1F98431c8aD98523631AE4a59f267346ea31F984', chain: 'ethereum' },

{ name: 'balancer', address: '0xBA12222222228d8Ba445958a75a0704d566BF2C8', chain: 'ethereum' },

// Add more for other chains

];

for (const protocol of protocols) {

const provider = providers[protocol.chain];

// Fetch liquidity data (simplified)

await pool.query(`

INSERT INTO markets (chain\_id, protocol, address, metadata)

VALUES ((SELECT id FROM chains WHERE name = $1), $2, $3, $4)

ON CONFLICT DO NOTHING

`, [protocol.chain, protocol.name, protocol.address, {}]);

}

}

// Simulate transaction

async function simulateTransaction(chain: string, contract: string, params: any) {

const provider = providers[chain];

const contractInstance = new ethers.Contract(contract, ['function executeOperation(address,uint256,uint256,address,bytes)'], provider);

const result = await contractInstance.callStatic.executeOperation(...params);

return { profit: result.profit, gas: result.gasEstimate };

}

// AI Module (simplified, rule-based with learning placeholder)

class FlashLoanAI {

async generateSmartContract(opportunity: any): Promise<string> {

// Placeholder: Generate Solidity code based on opportunity

return `// Generated contract for ${opportunity.protocol}\n${await import('./FlashLoanExecutor.sol')}`;

}

async optimizeStrategy(opportunities: any[]): Promise<any> {

// Rank by profit - (gas + fees)

return opportunities.sort((a, b) => (b.profit - (b.gas + b.fees)) - (a.profit - (a.gas + a.fees)))[0];

}

async handleError(error: any): Promise<string> {

// Placeholder: Analyze and fix errors

return `Error: ${error.message}. Suggested fix: Check gas limits or contract logic.`;

}

}

const ai = new FlashLoanAI();

// API Endpoints

app.get('/opportunities', async (req, res) => {

const { filter } = req.query;

const query = filter === 'zero\_cost'

? `SELECT \* FROM simulations WHERE expected\_profit\_wei > gas\_wei + flash\_fee\_wei`

: `SELECT \* FROM simulations`;

const result = await pool.query(query);

res.json(result.rows);

});

app.post('/execute', async (req, res) => {

const { chain, contract, params } = req.body;

const provider = providers[chain];

const signer = executorWallet.connect(provider);

const contractInstance = new ethers.Contract(contract, ['function executeOperation(address,uint256,uint256,address,bytes)'], signer);

const tx = await contractInstance.executeOperation(...params);

await tx.wait();

res.json({ txHash: tx.hash });

});

app.post('/transfer-profit', async (req, res) => {

const { token, amount, to } = req.body;

const provider = providers['ethereum']; // Default to Ethereum

const signer = new ethers.Wallet(process.env.EXECUTOR\_PRIVATE\_KEY!, provider);

const contract = new ethers.Contract('FlashLoanExecutor', ['function withdrawProfit(address,address,uint256)'], signer);

const tx = await contract.withdrawProfit(token, to, amount);

await tx.wait();

res.json({ txHash: tx.hash });

});

app.post('/ai-chat', async (req, res) => {

const { message } = req.body;

const response = await ai.handleError({ message });

res.json({ response });

});

// Network Toggle

let networkActive = false;

app.post('/toggle-network', (req, res) => {

networkActive = req.body.active;

res.json({ status: networkActive ? 'Active' : 'Inactive' });

});

// Start server

initDb().then(() => {

cron.schedule('\*/5 \* \* \* \*', indexOpportunities); // Run every 5 minutes

app.listen(3001, () => console.log('Server running on port 3001'));

});