```
// ==========================================================
// MAUAX DEPLOYMENT AND CONFIGURATION SCRIPTS
// ==========================================================

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "./MauaxFoundersNFT.sol";
import "./MauaxUtilityToken.sol";
import "./MauaxEnergyToken.sol";
import "./MauaxRecyclingToken.sol";
import "./MauaxSeedNFT.sol";
import "./InvestorVault.sol";
import "./MauaxSecurityTokenFactory.sol";
import "./MauaxDAOTreasury.sol";
import "./OracleEnergyData.sol";
import "./MauaxStakingSystem.sol";

/**
 * @title MAUAX Master Deployer
 * @notice Contrato para deploy coordenado de todo o ecossistema MAUAX
 * @dev Sequência: DEV → SEC → DEPLOY → POST-DEPLOY → MINT → DISTRIBUTION
 */
contract MauaxMasterDeployer is AccessControl {
    bytes32 public constant DEPLOYER_ROLE = keccak256("DEPLOYER_ROLE");

    // Deployed contract addresses
    struct DeployedContracts {
        address foundersNFT;
        address utilityToken;
        address energyToken;
        address recyclingToken;
        address seedNFT;
        address investorVault;
        address seedSale;
        address securityTokenFactory;
        address daoTreasury;
        address energyOracle;
        address stakingSystem;
        address pspIntegration;
        address crossChainBridge;
        address dexIntegration;
        address insuranceProtocol;
    }
```

```
DeployedContracts public contracts;
address public gnosisSafeAddress;
bool public deploymentCompleted;

enum DeploymentPhase {
    PREPARATION,
    CORE_CONTRACTS,
    SECURITY_TOKENS,
    INFRASTRUCTURE,
    CONFIGURATION,
    COMPLETED
}

DeploymentPhase public currentPhase = DeploymentPhase.PREPARATION;

event ContractDeployed(string contractName, address contractAddress);
event PhaseCompleted(DeploymentPhase phase);
event OwnershipTransferred(address contractAddress, address newOwner);

constructor() {
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(DEPLOYER_ROLE, msg.sender);
}

/**
 * @notice FASE 1: Deploy dos contratos principais
 */
function deployCoreContracts() external onlyRole(DEPLOYER_ROLE) {
    require(currentPhase == DeploymentPhase.PREPARATION, "Wrong phase");

    // 1. Deploy Founders NFT
    MauaxFoundersNFT foundersNFT = new MauaxFoundersNFT();
    contracts.foundersNFT = address(foundersNFT);
    emit ContractDeployed("MauaxFoundersNFT", contracts.foundersNFT);

    // 2. Deploy Utility Token
    MauaxUtilityToken utilityToken = new MauaxUtilityToken();
    contracts.utilityToken = address(utilityToken);
    emit ContractDeployed("MauaxUtilityToken", contracts.utilityToken);

    // 3. Deploy Energy Token
    MauaxEnergyToken energyToken = new MauaxEnergyToken();
    contracts.energyToken = address(energyToken);
    emit ContractDeployed("MauaxEnergyToken", contracts.energyToken);
    // 4. Deploy Recycling Token
    MauaxRecyclingToken recyclingToken = new MauaxRecyclingToken();
    contracts.recyclingToken = address(recyclingToken);
    emit ContractDeployed("MauaxRecyclingToken", contracts.recyclingToken);

    // 5. Deploy Seed NFT
    MauaxSeedNFT seedNFT = new MauaxSeedNFT();
    contracts.seedNFT = address(seedNFT);
    emit ContractDeployed("MauaxSeedNFT", contracts.seedNFT);

    // 6. Deploy Investor Vault
    InvestorVault investorVault = new InvestorVault(contracts.seedNFT);
    contracts.investorVault = address(investorVault);
    emit ContractDeployed("InvestorVault", contracts.investorVault);

    currentPhase = DeploymentPhase.CORE_CONTRACTS;
    emit PhaseCompleted(DeploymentPhase.CORE_CONTRACTS);
}

/**
 * @notice FASE 2: Deploy dos Security Tokens
 */
function deploySecurityTokens() external onlyRole(DEPLOYER_ROLE) {
    require(currentPhase == DeploymentPhase.CORE_CONTRACTS, "Wrong phase");

    // Deploy Security Token Factory
    MauaxSecurityTokenFactory factory = new MauaxSecurityTokenFactory();
    contracts.securityTokenFactory = address(factory);
    emit ContractDeployed("MauaxSecurityTokenFactory", contracts.securityTokenFactory);

    // Deploy all security tokens through factory
    factory.deployAllTokens();

    currentPhase = DeploymentPhase.SECURITY_TOKENS;
    emit PhaseCompleted(DeploymentPhase.SECURITY_TOKENS);
}

/**
 * @notice FASE 3: Deploy da infraestrutura
 */
function deployInfrastructure() external onlyRole(DEPLOYER_ROLE) {
    require(currentPhase == DeploymentPhase.SECURITY_TOKENS, "Wrong phase");

    // 1. DAO Treasury
    MauaxDAOTreasury treasury = new MauaxDAOTreasury();
```

```
        contracts.daoTreasury = address(treasury);
        emit ContractDeployed("MauaxDAOTreasury", contracts.daoTreasury);

        // 2. Energy Oracle
        OracleEnergyData oracle = new OracleEnergyData(contracts.energyToken);
        contracts.energyOracle = address(oracle);
        emit ContractDeployed("OracleEnergyData", contracts.energyOracle);

        // 3. Staking System
        MauaxStakingSystem staking = new MauaxStakingSystem(contracts.utilityToken);
        contracts.stakingSystem = address(staking);
        emit ContractDeployed("MauaxStakingSystem", contracts.stakingSystem);

        currentPhase = DeploymentPhase.INFRASTRUCTURE;
        emit PhaseCompleted(DeploymentPhase.INFRASTRUCTURE);
    }

    /**
     * @notice FASE 4: Configuração e transferência de ownership
     */
    function configureContracts(address _gnosisSafeAddress) external
onlyRole(DEPLOYER_ROLE) {
        require(currentPhase == DeploymentPhase.INFRASTRUCTURE, "Wrong phase");
        require(_gnosisSafeAddress != address(0), "Invalid Gnosis Safe address");

        gnosisSafeAddress = _gnosisSafeAddress;

        // Transfer ownership of all contracts to Gnosis Safe
        _transferOwnership(contracts.foundersNFT, "MauaxFoundersNFT");
        _transferOwnership(contracts.utilityToken, "MauaxUtilityToken");
        _transferOwnership(contracts.energyToken, "MauaxEnergyToken");
        _transferOwnership(contracts.recyclingToken, "MauaxRecyclingToken");
        _transferOwnership(contracts.seedNFT, "MauaxSeedNFT");

        // Configure role-based access for other contracts
        _configureAccessControl();

        currentPhase = DeploymentPhase.CONFIGURATION;
        emit PhaseCompleted(DeploymentPhase.CONFIGURATION);
    }

    /**
     * @notice FASE 5: Emissão inicial dos tokens
     */
    function mintInitialTokens() external onlyRole(DEPLOYER_ROLE) {
```

```
require(currentPhase == DeploymentPhase.CONFIGURATION, "Wrong phase");
require(gnosisSafeAddress != address(0), "Gnosis Safe not set");

// This function would be called BY the Gnosis Safe, not by the deployer
// The deployer just marks the phase as ready for minting

currentPhase = DeploymentPhase.COMPLETED;
deploymentCompleted = true;
emit PhaseCompleted(DeploymentPhase.
```