```
// ========================================================
// MAUAX DEPLOYMENT AND CONFIGURATION SCRIPTS - COMPLETED
// ========================================================
// 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";
import "./MauaxSeedSale.sol";
import "./MauaxPSPIntegration.sol";
import "./MauaxCrossChainBridge.sol";
import "./MauaxDEXIntegration.sol";
import "./MauaxInsuranceProtocol.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);
event DeploymentFinalized(address gnosisSafe, uint256 timestamp);

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);

        // 7. Deploy Seed Sale
        MauaxSeedSale seedSale = new MauaxSeedSale(contracts.seedNFT,
contracts.investorVault);
        contracts.seedSale = address(seedSale);
        emit ContractDeployed("MauaxSeedSale", contracts.seedSale);

        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);

    // 4. PSP Integration
    MauaxPSPIntegration psp = new MauaxPSPIntegration(
        contracts.utilityToken,
        contracts.recyclingToken,
        contracts.energyToken
    );
    contracts.pspIntegration = address(psp);
    emit ContractDeployed("MauaxPSPIntegration", contracts.pspIntegration);

    // 5. Cross Chain Bridge
    MauaxCrossChainBridge bridge = new MauaxCrossChainBridge();
    contracts.crossChainBridge = address(bridge);
    emit ContractDeployed("MauaxCrossChainBridge", contracts.crossChainBridge);

    // 6. Insurance Protocol
    MauaxInsuranceProtocol insurance = new MauaxInsuranceProtocol();
    contracts.insuranceProtocol = address(insurance);
    emit ContractDeployed("MauaxInsuranceProtocol", contracts.insuranceProtocol);

    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();

        // Setup initial connections between contracts
        _setupContractConnections();

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

    /**
    * @notice FASE 5: Finalização e emissão inicial
    */
    function finalizeDeployment() external onlyRole(DEPLOYER_ROLE) {
        require(currentPhase == DeploymentPhase.CONFIGURATION, "Wrong phase");
        require(gnosisSafeAddress != address(0), "Gnosis Safe not set");

        // Mint initial NFTs to treasury
        MauaxFoundersNFT(contracts.foundersNFT).mintAllToTreasury();

        // Mint SEED NFT to sale contract
        MauaxSeedNFT(contracts.seedNFT).mintToSaleContract(contracts.seedSale);

        // Setup initial energy oracle data
        _setupEnergyOracle();
    // Configure security token factory permissions
    _configureSecurityTokens();

    currentPhase = DeploymentPhase.COMPLETED;
    deploymentCompleted = true;

    emit PhaseCompleted(DeploymentPhase.COMPLETED);
    emit DeploymentFinalized(gnosisSafeAddress, block.timestamp);
}

/**
* @dev Transfer ownership of a contract to Gnosis Safe
*/
function _transferOwnership(address contractAddress, string memory contractName) internal {
    try Ownable(contractAddress).transferOwnership(gnosisSafeAddress) {
        emit OwnershipTransferred(contractAddress, gnosisSafeAddress);
    } catch {
        // For AccessControl contracts, transfer admin role
        try AccessControl(contractAddress).grantRole(
            AccessControl(contractAddress).DEFAULT_ADMIN_ROLE(),
            gnosisSafeAddress
        ) {
            emit OwnershipTransferred(contractAddress, gnosisSafeAddress);
        } catch {
            // Log error but continue deployment
        }
    }
}

/**
* @dev Configure access control for ecosystem contracts
*/
function _configureAccessControl() internal {
    // Grant necessary roles to contracts

    // Energy Oracle can mint energy tokens
    MauaxEnergyToken(contracts.energyToken).grantRole(
        keccak256("ORACLE_ROLE"),
        contracts.energyOracle
    );

    // Staking system can mint utility tokens for rewards
    MauaxUtilityToken(contracts.utilityToken).authorizeMinter(contracts.stakingSystem);

    // PSP can process all token types
```

```
        MauaxUtilityToken(contracts.utilityToken).authorizeMinter(contracts.pspIntegration);

        // Cross-chain bridge can handle tokens

MauaxCrossChainBridge(contracts.crossChainBridge).addSupportedToken(contracts.utilityToken
);

MauaxCrossChainBridge(contracts.crossChainBridge).addSupportedToken(contracts.energyToke
n);

MauaxCrossChainBridge(contracts.crossChainBridge).addSupportedToken(contracts.recyclingTo
ken);
    }

    /**
    * @dev Setup connections between contracts
    */
    function _setupContractConnections() internal {
        // Connect Oracle to Energy Token
        OracleEnergyData(contracts.energyOracle).grantRole(
            keccak256("DATA_PROVIDER_ROLE"),
            gnosisSafeAddress
        );

        // Setup security token factory permissions
        MauaxSecurityTokenFactory(contracts.securityTokenFactory).grantRole(
            keccak256("FACTORY_MANAGER_ROLE"),
            gnosisSafeAddress
        );
    }

    /**
    * @dev Setup initial energy oracle configuration
    */
    function _setupEnergyOracle() internal {
        // Add initial data providers - this would be done by Gnosis Safe later
        // Just setup the basic structure here
    }

    /**
    * @dev Configure security token factory and initial tokens
    */
    function _configureSecurityTokens() internal {
        MauaxSecurityTokenFactory factory =
MauaxSecurityTokenFactory(contracts.securityTokenFactory);
```

```
        // Transfer security token ownerships to Gnosis Safe
        address[] memory deployedTokens = factory.getDeployedTokens();
        for (uint256 i = 0; i < deployedTokens.length; i++) {
            factory.transferTokenOwnership(deployedTokens[i], gnosisSafeAddress);
        }
    }

    /**
    * @notice Emergency function to transfer deployer role
    */
    function transferDeployerRole(address newDeployer) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        require(newDeployer != address(0), "Invalid address");
        _grantRole(DEPLOYER_ROLE, newDeployer);
        _revokeRole(DEPLOYER_ROLE, msg.sender);
    }

    /**
    * @notice Get all deployed contract addresses
    */
    function getDeployedContracts() external view returns (DeployedContracts memory) {
        return contracts;
    }

    /**
    * @notice Check if deployment is complete
    */
    function isDeploymentComplete() external view returns (bool) {
        return deploymentCompleted && currentPhase == DeploymentPhase.COMPLETED;
    }

    /**
    * @notice Get deployment progress
    */
    function getDeploymentProgress() external view returns (
        DeploymentPhase phase,
        uint256 progressPercentage,
        bool isComplete
    ) {
        uint256 percentage = (uint256(currentPhase) * 100) /
uint256(DeploymentPhase.COMPLETED);
        return (currentPhase, percentage, deploymentCompleted);
    }
}
```