```
// =======================================================
// MAUAX SECURITY TOKENS - SPECIFIC IMPLEMENTATIONS
// =======================================================

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

import "./MauaxSecurityToken.sol";

// =======================================================
// 1. MAUAX-S SOLAR - ENERGIA JUSTA MAUÁ
// =======================================================

/**
 * @title MAUAX Solar Security Token
 * @notice Security token para investimento no programa Energia Justa Mauá
 * @dev CAPEX: US$ 700 milhões | TIR: 18% a.a. | Perfil: Conservador
 */
contract MauaxSolarToken is MauaxSecurityToken {
    uint256 public constant SOLAR_CAPEX = 700_000_000; // US$ 700 milhões
    uint256 public constant EXPECTED_TIR = 1800; // 18% em basis points
    uint256 public constant TOKEN_SUPPLY = 700_000; // 700.000 tokens

    struct SolarProject {
        string location;
        uint256 capacity; // em kW
        uint256 expectedGeneration; // kWh/ano
        bool operational;
        uint256 installationDate;
    }

    mapping(uint256 => SolarProject) public solarProjects;
    uint256 public projectCounter;
    uint256 public totalInstalledCapacity;
    uint256 public totalEnergyGenerated;

    event SolarProjectAdded(uint256 indexed projectId, string location, uint256 capacity);
    event EnergyGenerated(uint256 indexed projectId, uint256 amount, uint256 timestamp);
    event DividendsFromEnergy(uint256 totalRevenue, uint256 dividendPool);

    constructor() MauaxSecurityToken(
        "MAUAX Solar Investment Token",
        "MAUAX-S",
        TOKEN_SUPPLY,
        SOLAR_CAPEX,
        EXPECTED_TIR,
        "Investimento em energia solar distribuida - Programa Energia Justa Maua"
    ) {}

    function addSolarProject(
        string memory location,
        uint256 capacity,
        uint256 expectedGeneration
    ) external onlyRole(ADMIN_ROLE) returns (uint256) {
        projectCounter++;
        solarProjects[projectCounter] = SolarProject({
            location: location,
            capacity: capacity,
            expectedGeneration: expectedGeneration,
            operational: false,
            installationDate: 0
        });

        totalInstalledCapacity += capacity;
        emit SolarProjectAdded(projectCounter, location, capacity);
        return projectCounter;
    }

    function markProjectOperational(uint256 projectId) external onlyRole(ADMIN_ROLE) {
        SolarProject storage project = solarProjects[projectId];
        require(!project.operational, "Already operational");

        project.operational = true;
        project.installationDate = block.timestamp;
    }

    function recordEnergyGeneration(uint256 projectId, uint256 amount) external
onlyRole(ADMIN_ROLE) {
        require(solarProjects[projectId].operational, "Project not operational");

        totalEnergyGenerated += amount;
        emit EnergyGenerated(projectId, amount, block.timestamp);

        // Calculate revenue and distribute dividends
        _calculateAndDistributeDividends(amount);
    }

    function _calculateAndDistributeDividends(uint256 energyAmount) internal {
        // Assuming average price of R$ 0.30/kWh
        uint256 revenue = energyAmount * 30; // Revenue in cents
        uint256 dividendPool = (revenue * 70) / 100; // 70% to token holders

        emit DividendsFromEnergy(revenue, dividendPool);
        // Actual dividend distribution would be implemented here
    }

    function getProjectInfo(uint256 projectId) external view returns (
        string memory location,
        uint256 capacity,
        uint256 expectedGeneration,
        bool operational,
        uint256 installationDate
    ) {
        SolarProject storage project = solarProjects[projectId];
        return (
            project.location,
            project.capacity,
            project.expectedGeneration,
            project.operational,
            project.installationDate
        );
    }
}

// ============================================================
// 2. MAUAX-B BIOPOLO - ECONOMIA CIRCULAR
// ============================================================

/**
 * @title MAUAX Biopolo Security Token
 * @notice Security token para investimento no Biopolo MAUAX
 * @dev CAPEX: US$ 1.2 bilhões | TIR: 25% a.a. | Perfil: Crescimento
 */
contract MauaxBiopoloToken is MauaxSecurityToken {
    uint256 public constant BIOPOLO_CAPEX = 1_200_000_000; // US$ 1.2 bilhões
    uint256 public constant EXPECTED_TIR = 2500; // 25% em basis points
    uint256 public constant TOKEN_SUPPLY = 120_000; // 120.000 tokens

    struct BiochemicalProduct {
        string name;
        uint256 productionCapacity; // tons/year
        uint256 marketPrice; // USD per ton
        bool inProduction;
        uint256 totalProduced;
    }
```

```
mapping(uint256 => BiochemicalProduct) public products;
uint256 public productCounter;
uint256 public totalRecycledMaterial;
uint256 public totalBiochemicalRevenue;

mapping(address => bool) public authorizedCooperatives;
mapping(string => uint256) public materialPrices; // material -> price per kg

event ProductAdded(uint256 indexed productId, string name, uint256 capacity);
event MaterialProcessed(string materialType, uint256 amount, uint256 revenue);
event BiochemicalProduced(uint256 indexed productId, uint256 amount, uint256 revenue);
event CooperativeRevenue(address indexed cooperative, uint256 amount);

constructor() MauaxSecurityToken(
    "MAUAX Biopolo Investment Token",
    "MAUAX-B",
    TOKEN_SUPPLY,
    BIOPOLO_CAPEX,
    EXPECTED_TIR,
    "Investimento em complexo industrial de economia circular e bioquimicos"
) {
    // Initialize material prices (in cents per kg)
    materialPrices["plastico"] = 80;
    materialPrices["papel"] = 45;
    materialPrices["metal"] = 150;
    materialPrices["vidro"] = 30;
    materialPrices["organico"] = 25;
}

function addBiochemicalProduct(
    string memory name,
    uint256 productionCapacity,
    uint256 marketPrice
) external onlyRole(ADMIN_ROLE) returns (uint256) {
    productCounter++;
    products[productCounter] = BiochemicalProduct({
        name: name,
        productionCapacity: productionCapacity,
        marketPrice: marketPrice,
        inProduction: false,
        totalProduced: 0
    });

    emit ProductAdded(productCounter, name, productionCapacity);
    return productCounter;
}

function processRecycledMaterial(
    string memory materialType,
    uint256 amount,
    address cooperative
) external onlyRole(ADMIN_ROLE) {
    require(authorizedCooperatives[cooperative], "Cooperative not authorized");
    require(materialPrices[materialType] > 0, "Material type not supported");

    uint256 revenue = amount * materialPrices[materialType];
    totalRecycledMaterial += amount;
    totalBiochemicalRevenue += revenue;

    // 30% goes to cooperative, 70% to token holders
    uint256 cooperativeShare = (revenue * 30) / 100;

    emit MaterialProcessed(materialType, amount, revenue);
    emit CooperativeRevenue(cooperative, cooperativeShare);

    // Transfer cooperative share (implementation would use actual payment)
    _distributeDividends(revenue - cooperativeShare);
}

function recordBiochemicalProduction(
    uint256 productId,
    uint256 amount
) external onlyRole(ADMIN_ROLE) {
    BiochemicalProduct storage product = products[productId];
    require(product.inProduction, "Product not in production");

    uint256 revenue = amount * product.marketPrice;
    product.totalProduced += amount;
    totalBiochemicalRevenue += revenue;

    emit BiochemicalProduced(productId, amount, revenue);
    _distributeDividends(revenue);
}

function startProductionLine(uint256 productId) external onlyRole(ADMIN_ROLE) {
    products[productId].inProduction = true;
}

function authorizeCooperative(address cooperative) external onlyRole(ADMIN_ROLE) {
        authorizedCooperatives[cooperative] = true;
    }

    function updateMaterialPrice(string memory materialType, uint256 newPrice) external
onlyRole(ADMIN_ROLE) {
        materialPrices[materialType] = newPrice;
    }

    function _distributeDividends(uint256 revenue) internal {
        // Dividend distribution logic would be implemented here
        emit DividendsDistributed(revenue);
    }
}

// ============================================================
// 3. MAUAX-D DATACENTER - BIO DATA CLOUD
// ============================================================

/**
 * @title MAUAX DataCenter Security Token
 * @notice Security token para investimento no Bio Data Cloud
 * @dev CAPEX: US$ 500 milhões | TIR: 30% a.a. | Perfil: Alto Crescimento
 */
contract MauaxDataCenterToken is MauaxSecurityToken {
    uint256 public constant DATACENTER_CAPEX = 500_000_000; // US$ 500 milhões
    uint256 public constant EXPECTED_TIR = 3000; // 30% em basis points
    uint256 public constant TOKEN_SUPPLY = 100_000; // 100.000 tokens

    struct ComputeNode {
        uint256 nodeId;
        string nodeType; // "GPU", "CPU", "STORAGE"
        uint256 computePower; // TFLOPS or GB
        uint256 hourlyRate; // USD per hour
        bool operational;
        uint256 totalUptime;
        uint256 totalRevenue;
    }

    struct AIWorkload {
        uint256 workloadId;
        address client;
        string workloadType;
        uint256 computeHours;
        uint256 totalCost;
        uint256 startTime;
        uint256 endTime;
        bool completed;
    }

    mapping(uint256 => ComputeNode) public computeNodes;
    mapping(uint256 => AIWorkload) public aiWorkloads;
    mapping(address => bool) public authorizedClients;

    uint256 public nodeCounter;
    uint256 public workloadCounter;
    uint256 public totalComputeRevenue;
    uint256 public totalEnergyConsumed; // kWh

    event NodeAdded(uint256 indexed nodeId, string nodeType, uint256 computePower);
    event WorkloadStarted(uint256 indexed workloadId, address indexed client, string
workloadType);
    event WorkloadCompleted(uint256 indexed workloadId, uint256 revenue);
    event EnergyConsumed(uint256 amount, uint256 cost);

    constructor() MauaxSecurityToken(
        "MAUAX DataCenter Investment Token",
        "MAUAX-D",
        TOKEN_SUPPLY,
        DATACENTER_CAPEX,
        EXPECTED_TIR,
        "Investimento em Bio Data Cloud - Datacenter TIER IV com IA/HPC"
    ) {}

    function addComputeNode(
        string memory nodeType,
        uint256 computePower,
        uint256 hourlyRate
    ) external onlyRole(ADMIN_ROLE) returns (uint256) {
        nodeCounter++;
        computeNodes[nodeCounter] = ComputeNode({
            nodeId: nodeCounter,
            nodeType: nodeType,
            computePower: computePower,
            hourlyRate: hourlyRate,
            operational: false,
            totalUptime: 0,
            totalRevenue: 0
        });

        emit NodeAdded(nodeCounter, nodeType, computePower);
        return nodeCounter;
    }

    function startAIWorkload(
        address client,
        string memory workloadType,
        uint256 estimatedHours,
        uint256[] memory nodeIds
    ) external onlyRole(ADMIN_ROLE) returns (uint256) {
        require(authorizedClients[client], "Client not authorized");

        // Calculate total cost
        uint256 totalCost = 0;
        for (uint256 i = 0; i < nodeIds.length; i++) {
            ComputeNode storage node = computeNodes[nodeIds[i]];
            require(node.operational, "Node not operational");
            totalCost += node.hourlyRate * estimatedHours;
        }

        workloadCounter++;
        aiWorkloads[workloadCounter] = AIWorkload({
            workloadId: workloadCounter,
            client: client,
            workloadType: workloadType,
            computeHours: estimatedHours,
            totalCost: totalCost,
            startTime: block.timestamp,
            endTime: 0,
            completed: false
        });

        emit WorkloadStarted(workloadCounter, client, workloadType);
        return workloadCounter;
    }

    function completeWorkload(
        uint256 workloadId,
        uint256 actualHours,
        uint256 energyConsumed
    ) external onlyRole(ADMIN_ROLE) {
        AIWorkload storage workload = aiWorkloads[workloadId];
        require(!workload.completed, "Already completed");

        workload.endTime = block.timestamp;
        workload.completed = true;
```

```
        workload.computeHours = actualHours;

        totalComputeRevenue += workload.totalCost;
        totalEnergyConsumed += energyConsumed;

        emit WorkloadCompleted(workloadId, workload.totalCost);
        emit EnergyConsumed(energyConsumed, energyConsumed * 30); // Assuming 30
cents/kWh

        // Distribute revenue to token holders
        _distributeDividends(workload.totalCost);
    }

    function setNodeOperational(uint256 nodeId, bool operational) external
onlyRole(ADMIN_ROLE) {
        computeNodes[nodeId].operational = operational;
    }

    function authorizeClient(address client) external onlyRole(ADMIN_ROLE) {
        authorizedClients[client] = true;
    }

    function updateNodeRate(uint256 nodeId, uint256 newRate) external onlyRole(ADMIN_ROLE)
{
        computeNodes[nodeId].hourlyRate = newRate;
    }

    function getDataCenterStats() external view returns (
        uint256 totalNodes,
        uint256 activeWorkloads,
        uint256 totalRevenue,
        uint256 energyEfficiency
    ) {
        uint256 activeCount = 0;
        for (uint256 i = 1; i <= workloadCounter; i++) {
            if (!aiWorkloads[i].completed) {
                activeCount++;
            }
        }

        uint256 efficiency = totalEnergyConsumed > 0 ? (totalComputeRevenue * 1000) /
totalEnergyConsumed : 0;

        return (nodeCounter, activeCount, totalComputeRevenue, efficiency);
    }
    function _distributeDividends(uint256 revenue) internal {
        // 80% to token holders, 20% for operations
        uint256 dividendAmount = (revenue * 80) / 100;
        emit DividendsDistributed(dividendAmount);
    }
}

// =============================================================
// 4. MAUAX-T TOWER - HUB DE INOVAÇÃO
// =============================================================

/**
 * @title MAUAX Tower Security Token
 * @notice Security token para investimento na Mauá Tower
 * @dev CAPEX: US$ 200 milhões | TIR: 15% a.a. | Perfil: Conservador
 */
contract MauaxTowerToken is MauaxSecurityToken {
    uint256 public constant TOWER_CAPEX = 200_000_000; // US$ 200 milhões
    uint256 public constant EXPECTED_TIR = 1500; // 15% em basis points
    uint256 public constant TOKEN_SUPPLY = 100_000; // 100.000 tokens

    struct PropertyUnit {
        uint256 unitId;
        string unitType; // "OFFICE", "RETAIL", "COWORKING", "EVENT"
        uint256 area; // m²
        uint256 monthlyRent; // USD
        address tenant;
        uint256 leaseStart;
        uint256 leaseEnd;
        bool occupied;
    }

    struct StartupIncubation {
        uint256 startupId;
        string companyName;
        address founder;
        uint256 equity; // percentage
        uint256 investmentAmount;
        uint256 incubationStart;
        bool active;
        uint256 valuation;
    }

    mapping(uint256 => PropertyUnit) public propertyUnits;
    mapping(uint256 => StartupIncubation) public incubatedStartups;
    mapping(address => uint256) public tenantDeposits;

    uint256 public unitCounter;
    uint256 public startupCounter;
    uint256 public totalRentalRevenue;
    uint256 public totalIncubationRevenue;
    uint256 public occupancyRate; // in basis points

    event UnitAdded(uint256 indexed unitId, string unitType, uint256 area, uint256 rent);
    event UnitLeased(uint256 indexed unitId, address indexed tenant, uint256 duration);
    event StartupIncubated(uint256 indexed startupId, string companyName, uint256 investment);
    event RentCollected(uint256 indexed unitId, uint256 amount);
    event StartupExit(uint256 indexed startupId, uint256 exitValuation, uint256 returns);

    constructor() MauaxSecurityToken(
        "MAUAX Tower Investment Token",
        "MAUAX-T",
        TOKEN_SUPPLY,
        TOWER_CAPEX,
        EXPECTED_TIR,
        "Investimento em Maua Tower - Hub de inovacao e centro comercial"
    ) {}

    function addPropertyUnit(
        string memory unitType,
        uint256 area,
        uint256 monthlyRent
    ) external onlyRole(ADMIN_ROLE) returns (uint256) {
        unitCounter++;
        propertyUnits[unitCounter] = PropertyUnit({
            unitId: unitCounter,
            unitType: unitType,
            area: area,
            monthlyRent: monthlyRent,
            tenant: address(0),
            leaseStart: 0,
            leaseEnd: 0,
            occupied: false
        });

        emit UnitAdded(unitCounter, unitType, area, monthlyRent);
        return unitCounter;
    }
```

```
function leaseUnit(
    uint256 unitId,
    address tenant,
    uint256 leaseDurationMonths,
    uint256 deposit
) external onlyRole(ADMIN_ROLE) {
    PropertyUnit storage unit = propertyUnits[unitId];
    require(!unit.occupied, "Unit already occupied");
    require(tenant != address(0), "Invalid tenant");

    unit.tenant = tenant;
    unit.leaseStart = block.timestamp;
    unit.leaseEnd = block.timestamp + (leaseDurationMonths * 30 days);
    unit.occupied = true;

    tenantDeposits[tenant] = deposit;

    emit UnitLeased(unitId, tenant, leaseDurationMonths);
    _updateOccupancyRate();
}

function collectRent(uint256 unitId) external onlyRole(ADMIN_ROLE) {
    PropertyUnit storage unit = propertyUnits[unitId];
    require(unit.occupied, "Unit not occupied");
    require(block.timestamp <= unit.leaseEnd, "Lease expired");

    uint256 rentAmount = unit.monthlyRent;
    totalRentalRevenue += rentAmount;

    emit RentCollected(unitId, rentAmount);

    // Distribute 90% to token holders, 10% for maintenance
    uint256 dividendAmount = (rentAmount * 90) / 100;
    _distributeDividends(dividendAmount);
}

function incubateStartup(
    string memory companyName,
    address founder,
    uint256 equity,
    uint256 investmentAmount
) external onlyRole(ADMIN_ROLE) returns (uint256) {
    startupCounter++;
    incubatedStartups[startupCounter] = StartupIncubation({
        startupId: startupCounter,
            companyName: companyName,
            founder: founder,
            equity: equity,
            investmentAmount: investmentAmount,
            incubationStart: block.timestamp,
            active: true,
            valuation: investmentAmount * 100 / equity // Initial valuation
        });

        totalIncubationRevenue += investmentAmount;

        emit StartupIncubated(startupCounter, companyName, investmentAmount);
        return startupCounter;
    }

    function exitStartup(
        uint256 startupId,
        uint256 exitValuation
    ) external onlyRole(ADMIN_ROLE) {
        StartupIncubation storage startup = incubatedStartups[startupId];
        require(startup.active, "Startup not active");

        startup.active = false;
        startup.valuation = exitValuation;

        // Calculate returns based on equity held
        uint256 returns = (exitValuation * startup.equity) / 100;
        uint256 profit = returns > startup.investmentAmount ? returns - startup.investmentAmount :
0;

        emit StartupExit(startupId, exitValuation, returns);

        if (profit > 0) {
            // Distribute profits to token holders
            _distributeDividends(profit);
        }
    }

    function terminateLease(uint256 unitId) external onlyRole(ADMIN_ROLE) {
        PropertyUnit storage unit = propertyUnits[unitId];
        require(unit.occupied, "Unit not occupied");

        delete tenantDeposits[unit.tenant];
        unit.tenant = address(0);
        unit.leaseStart = 0;
    unit.leaseEnd = 0;
    unit.occupied = false;

    _updateOccupancyRate();
}

function _updateOccupancyRate() internal {
    uint256 occupiedUnits = 0;
    for (uint256 i = 1; i <= unitCounter; i++) {
        if (propertyUnits[i].occupied) {
            occupiedUnits++;
        }
    }
    occupancyRate = unitCounter > 0 ? (occupiedUnits * 10000) / unitCounter : 0;
}

function getTowerStats() external view returns (
    uint256 totalUnits,
    uint256 occupiedUnits,
    uint256 currentOccupancyRate,
    uint256 monthlyRevenue,
    uint256 activeStartups
) {
    uint256 occupied = 0;
    uint256 monthlyRev = 0;
    uint256 activeCount = 0;

    for (uint256 i = 1; i <= unitCounter; i++) {
        if (propertyUnits[i].occupied) {
            occupied++;
            monthlyRev += propertyUnits[i].monthlyRent;
        }
    }

    for (uint256 i = 1; i <= startupCounter; i++) {
        if (incubatedStartups[i].active) {
            activeCount++;
        }
    }

    return (unitCounter, occupied, occupancyRate, monthlyRev, activeCount);
}

function _distributeDividends(uint256 amount) internal {
    emit DividendsDistributed(amount);
```

```
        // Actual dividend distribution implementation would go here
    }
}

// ============================================================
// 5. SECURITY TOKEN FACTORY - DEPLOYMENT MANAGER
// ============================================================

/**
 * @title MAUAX Security Token Factory
 * @notice Factory contract para deploy dos Security Tokens
 * @dev Gerencia a criação e configuração de todos os tokens de investimento
 */
contract MauaxSecurityTokenFactory is AccessControl {
    bytes32 public constant FACTORY_MANAGER_ROLE =
keccak256("FACTORY_MANAGER_ROLE");

    address[] public deployedTokens;
    mapping(string => address) public tokensBySymbol;
    mapping(address => bool) public isAuthorizedToken;

    event TokenDeployed(address indexed tokenAddress, string symbol, string name, uint256
supply);

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

    function deploySolarToken() external onlyRole(FACTORY_MANAGER_ROLE) returns
(address) {
        MauaxSolarToken solarToken = new MauaxSolarToken();
        address tokenAddress = address(solarToken);

        deployedTokens.push(tokenAddress);
        tokensBySymbol["MAUAX-S"] = tokenAddress;
        isAuthorizedToken[tokenAddress] = true;

        // Transfer ownership to this factory for initial setup
        solarToken.grantRole(solarToken.DEFAULT_ADMIN_ROLE(), address(this));

        emit TokenDeployed(tokenAddress, "MAUAX-S", "MAUAX Solar Investment Token",
700000);
        return tokenAddress;
    }
```

```
   function deployBiopoloToken() external onlyRole(FACTORY_MANAGER_ROLE) returns
(address) {
        MauaxBiopoloToken biopoloToken = new MauaxBiopoloToken();
        address tokenAddress = address(biopoloToken);

        deployedTokens.push(tokenAddress);
        tokensBySymbol["MAUAX-B"] = tokenAddress;
        isAuthorizedToken[tokenAddress] = true;

        biopoloToken.grantRole(biopoloToken.DEFAULT_ADMIN_ROLE(), address(this));

        emit TokenDeployed(tokenAddress, "MAUAX-B", "MAUAX Biopolo Investment Token",
120000);
        return tokenAddress;
   }

   function deployDataCenterToken() external onlyRole(FACTORY_MANAGER_ROLE) returns
(address) {
        MauaxDataCenterToken dataCenterToken = new MauaxDataCenterToken();
        address tokenAddress = address(dataCenterToken);

        deployedTokens.push(tokenAddress);
        tokensBySymbol["MAUAX-D"] = tokenAddress;
        isAuthorizedToken[tokenAddress] = true;

        dataCenterToken.grantRole(dataCenterToken.DEFAULT_ADMIN_ROLE(), address(this));

        emit TokenDeployed(tokenAddress, "MAUAX-D", "MAUAX DataCenter Investment Token",
100000);
        return tokenAddress;
   }

   function deployTowerToken() external onlyRole(FACTORY_MANAGER_ROLE) returns
(address) {
        MauaxTowerToken towerToken = new MauaxTowerToken();
        address tokenAddress = address(towerToken);

        deployedTokens.push(tokenAddress);
        tokensBySymbol["MAUAX-T"] = tokenAddress;
        isAuthorizedToken[tokenAddress] = true;

        towerToken.grantRole(towerToken.DEFAULT_ADMIN_ROLE(), address(this));
        emit TokenDeployed(tokenAddress, "MAUAX-T", "MAUAX Tower Investment Token",
100000);
        return tokenAddress;
    }

    function deployAllTokens() external onlyRole(FACTORY_MANAGER_ROLE) {
        deploySolarToken();
        deployBiopoloToken();
        deployDataCenterToken();
        deployTowerToken();
    }

    function getDeployedTokens() external view returns (address[] memory) {
        return deployedTokens;
    }

    function getTokenBySymbol(string memory symbol) external view returns (address) {
        return tokensBySymbol[symbol];
    }

    function getDeployedTokenCount() external view returns (uint256) {
        return deployedTokens.length;
    }

    function transferTokenOwnership(address tokenAddress, address newOwner) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        require(isAuthorizedToken[tokenAddress], "Token not authorized");
        MauaxSecurityToken(tokenAddress).grantRole(
            MauaxSecurityToken(tokenAddress).DEFAULT_ADMIN_ROLE(),
            newOwner
        );
    }
}
```