```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/TRC20/ITRC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/security/Pausable.sol";

contract PollCoin is ITRC20, Ownable, Pausable {
    uint256 private constant TOTAL_SUPPLY = 77_777_777 * (10 ** 18);
    uint256 private constant BASE_APY = 3; // 3% APY for 30 days
    uint256 private constant MID_APY = 5; // 5% APY for 90 days
    uint256 private constant HIGH_APY = 7; // 7% APY for 6 months
    uint256 private constant MAX_APY = 10; // 10% APY for 1 year
    uint256 private constant EARLY_UNSTAKE_PENALTY = 10; // 10% penalty for unstaking before 3
    uint256 private constant MID_UNSTAKE_PENALTY = 5; // 5% penalty for unstaking before 90 da

    constructor() ITRC20("PollCoin", "POL") {
        _mint(msg.sender, TOTAL_SUPPLY); // All tokens minted at deployment
    }

    // Staking Mechanism with Tiered Rewards and Penalties
    mapping(address => uint256) public stakedBalance;
    mapping(address => uint256) public stakingTimestamp;

    event Staked(address indexed user, uint256 amount);
    event Unstaked(address indexed user, uint256 amount, uint256 penalty);
    event RewardsClaimed(address indexed user, uint256 amount);

    function stake(uint256 amount) external whenNotPaused {
        require(balanceOf(msg.sender) >= amount, "Insufficient POL balance");
        require(amount > 0, "Cannot stake zero tokens");

        _transfer(msg.sender, address(this), amount);
        stakedBalance[msg.sender] += amount;
        stakingTimestamp[msg.sender] = block.timestamp;

        emit Staked(msg.sender, amount);
    }

    function unstake(uint256 amount) external whenNotPaused {
        require(stakedBalance[msg.sender] >= amount, "Insufficient staked balance");
        require(amount > 0, "Cannot unstake zero tokens");

        uint256 timeStaked = block.timestamp - stakingTimestamp[msg.sender];
```

```
    uint256 penalty = 0;

    if (timeStaked < 30 days) {
        penalty = (amount * EARLY_UNSTAKE_PENALTY) / 100;
    } else if (timeStaked < 90 days) {
        penalty = (amount * MID_UNSTAKE_PENALTY) / 100;
    }

    amount -= penalty;
    stakedBalance[msg.sender] -= (amount + penalty);
    _transfer(address(this), msg.sender, amount);

    emit Unstaked(msg.sender, amount, penalty);
}

function claimRewards() external whenNotPaused {
    uint256 rewards = calculateRewards(msg.sender);
    require(rewards > 0, "No rewards available");

    _transfer(address(this), msg.sender, rewards);
    emit RewardsClaimed(msg.sender, rewards);
}

function calculateRewards(address user) public view returns (uint256) {
    uint256 timeStaked = block.timestamp - stakingTimestamp[user];
    uint256 userStake = stakedBalance[user];
    uint256 apy;

    if (timeStaked >= 365 days) {
        apy = MAX_APY;
    } else if (timeStaked >= 180 days) {
        apy = HIGH_APY;
    } else if (timeStaked >= 90 days) {
        apy = MID_APY;
    } else {
        apy = BASE_APY;
    }

    return (userStake * apy * timeStaked) / (365 days * 100);
}

// Emergency function to pause staking in case of security concerns
function pause() external onlyOwner {
    _pause();
}
```

```
    function unpause() external onlyOwner {
        _unpause();
    }
}
```