```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;


import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";

import "@openzeppelin/contracts-
upgradeable/access/OwnableUpgradeable.sol";

import "@openzeppelin/contracts-
upgradeable/utils/cryptography/ECDSAUpgradeable.sol"; // Adicionar biblioteca
de criptografia

import "https://github.com/ethereum/EIPs/blob/master/EIPS/eip-712.md"; //
Exemplo de importação de biblioteca para EIP-712


contract CountryVoting is Initializable, ERC20, OwnableUpgradeable {

  uint256 public electionDate; // Data das eleições (timestamp)

  uint256 public lastBurnDate; // Última data de queima de tokens

  uint8 public constant DECIMALS = 6; // Permite a divisão do token em 1 milhão
de partes

  bool private locked; // Mutex para proteção contra reentrância

  uint256 public constant RATE_LIMIT = 100; // Número máximo de solicitações
por hora

  mapping(address => uint256) public requestCount;

  mapping(address => uint256) public lastRequestTime;


  AggregatorV3Interface internal priceFeed;

  AggregatorV3Interface internal fallbackPriceFeed; // Oráculo de fallback


  bool public tradingPaused; // Variável para pausar negociações
```

```
    uint256 public constant VOTING_AGE = 16; // Idade mínima para votar (em anos)

    struct Voter {
        bool voted;
        bool verified; // Verificação facial
        bool biometricallyVerified; // Verificação biométrica
        mapping(string => bool) votedForPosition; // Mapeamento para verificar se o eleitor votou para cada cargo
        string verificationCode;
        string email;
        string phoneNumber; // Número de telefone para SMS
        Vote[] votingHistory; // Histórico de votações
        string feedback; // Coleta de feedback dos eleitores
        string city; // Cidade do eleitor
        string state; // Estado do eleitor
        string name; // Nome do eleitor
        string address; // Endereço do eleitor
        string cpf; // CPF do eleitor
        string rg; // RG do eleitor
        uint256 birthdate; // Data de nascimento (timestamp)
        bool tokenIssued; // Indica se o token já foi emitido
        bytes32 pseudonym; // Pseudônimo do eleitor para proteção de privacidade
    }

    struct Vote {
        string position;
        address candidate;
        uint256 timestamp;
```

```
    }

    struct Candidate {
        string name;
        string position; // Cargo do candidato
        bool isValid;
        bool verified; // Verificação facial
        bool biometricallyVerified; // Verificação biométrica
        string biometricHash; // Adicionar hash biométrico para referência
        uint256 voteCount; // Contagem de votos do candidato
        uint256 termEndDate; // Data do fim do mandato atual
        string city; // Cidade do candidato
        string state; // Estado do candidato
        string address; // Endereço do candidato
        string cpf; // CPF do candidato
        string rg; // RG do candidato
        bool documentsVerified; // Indica se a documentação foi verificada
        bytes32 pseudonym; // Pseudônimo do candidato para proteção de
privacidade
    }

    struct Proposal {
        uint id;
        string description;
        uint256 voteCount;
        bool implemented;
        bool vetoed; // Indica se a proposta foi vetada
        address proposer; // Proponente da melhoria
```

```
    }

    address public owner;
    address[] public multiSigApprovers;
    mapping(address => Voter) public voters;
    mapping(address => Candidate) public candidates;
    mapping(uint => Proposal) public proposals;
    address[] public candidateList; // Lista de candidatos
    uint public proposalCount;
    mapping(uint8 => uint) public votes;
    bool public votingEnded;
    uint8 private multiSigCount;
    uint public totalVotes;

    event VoteCast(address indexed voter, string indexed position, uint8 indexed vote, string verificationCode);
    event CandidateRegistered(string name, string position, address indexed candidateAddress, string biometricHash);
    event VoterRegistered(address indexed voterAddress, string email, string phoneNumber, string verificationCode);
    event VotingEnded(address indexed ender);
    event ActionLogged(string actionType, address actor, uint timestamp);
    event InterferenceDetected(string description, string ipAddress, string location, uint timestamp);
    event TokensBurned(uint256 amount, uint256 timestamp); // Evento de queima de tokens
    event ElectionResults(address winner, uint256 voteCount); // Evento de resultado da eleição
    event SendNotification(string email, string phoneNumber, string message); // Evento para notificação
```

event CandidateDisqualified(address candidateAddress, string reason); // Evento de desqualificação do candidato

event ProposalCreated(uint proposalId, string description, address proposer); // Evento para criação de proposta

event ProposalVoted(uint proposalId, address voter); // Evento para voto em proposta

event ProposalVetoed(uint proposalId, address voter); // Evento para veto de proposta

event ProposalRewarded(uint proposalId, address proposer, uint256 rewardAmount); // Evento para recompensa de proposta aprovada

event AuditDataExported(address requester, string dataHash); // Evento de exportação de dados para auditoria

event SecurityIncidentDetected(string description, uint timestamp); // Evento para monitoramento de incidentes de segurança

event FacialVerificationRequested(address indexed user); // Evento para solicitar verificação facial

event FacialVerificationCompleted(address indexed user, bool success); // Evento para conclusão de verificação facial

event BiometricVerificationRequested(address indexed user); // Evento para solicitar verificação biométrica

event BiometricVerificationCompleted(address indexed user, bool success); // Evento para conclusão de verificação biométrica

event FeedbackCollected(address indexed voter, string feedback); // Evento para coleta de feedback

event TokenValueAdjusted(uint256 newValue); // Evento para ajuste de valor do token

event TieBreakVote(address indexed tokenHolder, string position); // Evento para voto de desempate

event DocumentVerificationRequested(address indexed candidate); // Evento para solicitar verificação de documentação

event DocumentVerificationCompleted(address indexed candidate, bool success); // Evento para conclusão de verificação de documentação

```
    event CrossChainCommunication(address indexed candidate, string indexed blockchain, string message); // Evento para comunicação entre blockchains

    event TradingPaused(uint256 timestamp); // Evento para pausar negociações

    event TradingResumed(uint256 timestamp); // Evento para retomar negociações

    event TokenIssued(address indexed voter, uint256 timestamp);

    event VoteDelegated(address indexed delegator, address indexed delegatee); // Evento para delegação de voto


    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can call this function.");
        _;
    }


    modifier onlyApprovers() {
        require(isApprover(msg.sender), "Only approvers can call this function.");
        _;
    }


    modifier hasNotVoted(string memory position) {
        require(!voters[msg.sender].votedForPosition[position], "You have already voted for this position.");
        _;
    }


    modifier votingActive() {
        require(!votingEnded, "Voting has ended.");
        _;
    }
```

```
    modifier registrationOpenForCandidates() {
        require(block.timestamp <= electionDate - 90 days, "Candidate registration is closed."); // 3
```