Solidity Compiler 8.16
Optimized / 200

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

/**
 * @title CryptoBroski Token Contract
 * @dev DISCLAIMER: This is an experimental token and is provided 'as is', without warranty of any kind.
 * The CryptoBroski Token (CBSKI) is a community-driven project. The developers and team behind this
 * project make no promises or guarantees regarding its value or stability. By interacting with this
 * contract, users acknowledge and accept that the token's use and value are subject to high risks,
 * including significant price volatility. Participation in this project should be considered a high-risk
 * activity, and users should do their own due diligence and consult with financial experts before
 * engaging in transactions involving the CryptoBroski Token. The team reserves the right to make
 * changes to the contract, but these changes will always be in pursuit of the community's benefit and
 * the token's sustainability.
 */

pragma solidity >=0.8.0 <0.9.0;


import "@openzeppelin/contracts@4.4.0/access/Ownable.sol";
import "@openzeppelin/contracts@4.4.0/security/ReentrancyGuard.sol";

contract CryptoBroski is Ownable, ReentrancyGuard {
    string public constant name = "CryptoBroski";
    string public constant symbol = "CBSKI";
    uint8 public constant decimals = 18;
    uint256 public totalSupply;

    uint256 public transactionTaxRate;
    uint256 public burnRate;
    bool public burnEnabled;


    address public taxWallet;
    address public taxExemptAddress;

    mapping(address => uint256) private _balances;
    mapping(address => mapping(address => uint256)) private _allowances;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
    event Burn(address indexed from, uint256 value);

    constructor(address _initialOwner, address _taxWallet) {
```

```
        totalSupply = 21000000 * (10 ** uint256(decimals));
        _balances[_initialOwner] = totalSupply;
        taxWallet = _taxWallet;
        transactionTaxRate = 100;
        burnRate = 100;
        burnEnabled = false;
    }

    function setTaxExemptAddress(address _taxExemptAddress) public onlyOwner {
        taxExemptAddress = _taxExemptAddress;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function transfer(address to, uint256 value) public nonReentrant returns (bool) {
        require(to != address(0), "Invalid address broski");
        _transfer(msg.sender, to, value);
        return true;
    }

    function _transfer(address from, address to, uint256 value) internal {
        require(_balances[from] >= value, "Insufficient balance broski");

        uint256 transactionTax = 0;
        uint256 burnAmount = 0;

        // Check if the destination address is 0x000000000000000000000000000000000000dEaD
or 0x0000000000000000000000000000000000000000
        if (to == address(0) || to == address(0x000000000000000000000000000000000000dEaD))
{
            // Burn all tokens in this case
            burnAmount = value;
        } else if (from != owner() && from != taxExemptAddress) {
            transactionTax = value * (transactionTaxRate) / (10000);
            if (burnEnabled) {
                burnAmount = value * (burnRate) / (10000);
            }
        }

        uint256 valueAfterTaxAndBurn = value - (transactionTax) - (burnAmount);
        _balances[from] = _balances[from] - (value);

        if (burnAmount > 0) {
            totalSupply = totalSupply - (burnAmount);
            emit Burn(from, burnAmount);
        }

        _balances[to] = _balances[to] + (valueAfterTaxAndBurn);
        emit Transfer(from, to, valueAfterTaxAndBurn);

        if (transactionTax > 0) {
            _balances[taxWallet] = _balances[taxWallet] + (transactionTax);
```

```
        }
    }

    function transferFrom(address from, address to, uint256 value) public nonReentrant returns
(bool) {
        require(from != address(0) && to != address(0), "Invalid address broski");
        require(_balances[from] >= value, "Insufficient balance broski");
        require(_allowances[from][msg.sender] >= value, "Allowance exceeded broski");

        _allowances[from][msg.sender] = _allowances[from][msg.sender] - (value);
        _transfer(from, to, value);
        return true;
    }

    function approve(address spender, uint256 value) public nonReentrant returns (bool) {
        _allowances[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        return true;
    }

    function burn(uint256 value) public {
        require(_balances[msg.sender] >= value, "Insufficient balance broski");
        _balances[msg.sender] = _balances[msg.sender] - (value);
        totalSupply = totalSupply - (value);
        emit Burn(msg.sender, value);
    }

    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }

    function setTransactionTaxRate(uint256 newRate) public onlyOwner {
        require(newRate <= 300, "Tax rate too high broski");
        transactionTaxRate = newRate;
    }

    function setBurnRate(uint256 newRate) public onlyOwner {
        require(newRate <= 300, "Burn rate too high broski");
        burnRate = newRate;
    }

    function setBurnEnabled(bool _burnEnabled) public onlyOwner {
        burnEnabled = _burnEnabled;
    }

    function setTaxWallet(address newTaxWallet) public onlyOwner {
        require(newTaxWallet != address(0), "Invalid address broski");
        taxWallet = newTaxWallet;
    }
}
```