```
/**
 *Submitted for verification at Etherscan.io on 2019-12-24
*/

pragma solidity ^0.5.15;

// File: @openzeppelin/contracts/utils/Address.sol

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is
returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
```

```
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     *
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

// File: @openzeppelin/contracts/introspection/ERC165Checker.sol

/**
 * @dev Library used to query support of an interface declared via {IERC165}.
 *
 * Note that these functions return the actual result of the query: they do not
 * `revert` if an interface is not supported. It is up to the caller to decide
 * what to do in these cases.
 */
library ERC165Checker {
    // As per the EIP-165 spec, no interface should ever match 0xffffffff
    bytes4 private constant _INTERFACE_ID_INVALID = 0xffffffff;

    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;

    /**
     * @dev Returns true if `account` supports the {IERC165} interface,
     */
    function _supportsERC165(address account) internal view returns (bool) {
        // Any contract that implements ERC165 must explicitly indicate support of
        // InterfaceId_ERC165 and explicitly indicate non-support of InterfaceId_Invalid
        return _supportsERC165Interface(account, _INTERFACE_ID_ERC165) &&
            !_supportsERC165Interface(account, _INTERFACE_ID_INVALID);
    }

    /**
```

```
     * @dev Returns true if `account` supports the interface defined by
     * `interfaceId`. Support for {IERC165} itself is queried automatically.
     *
     * See {IERC165-supportsInterface}.
     */
    function _supportsInterface(address account, bytes4 interfaceId) internal view returns (bool)
{
        // query support of both ERC165 as per the spec and support of _interfaceId
        return _supportsERC165(account) &&
            _supportsERC165Interface(account, interfaceId);
    }

    /**
     * @dev Returns true if `account` supports all the interfaces defined in
     * `interfaceIds`. Support for {IERC165} itself is queried automatically.
     *
     * Batch-querying can lead to gas savings by skipping repeated checks for
     * {IERC165} support.
     *
     * See {IERC165-supportsInterface}.
     */
    function _supportsAllInterfaces(address account, bytes4[] memory interfaceIds) internal view
returns (bool) {
        // query support of ERC165 itself
        if (!_supportsERC165(account)) {
            return false;
        }

        // query support of each interface in _interfaceIds
        for (uint256 i = 0; i < interfaceIds.length; i++) {
            if (!_supportsERC165Interface(account, interfaceIds[i])) {
                return false;
            }
        }

        // all interfaces supported
        return true;
    }

    /**
     * @notice Query if a contract implements an interface, does not check ERC165 support
     * @param account The address of the contract to query for support of an interface
     * @param interfaceId The interface identifier, as specified in ERC-165
     * @return true if the contract at account indicates support of the interface with
     * identifier interfaceId, false otherwise
     * @dev Assumes that account contains a contract that supports ERC165, otherwise
     * the behavior of this method is undefined. This precondition can be checked
     * with the `supportsERC165` method in this library.
     * Interface identification is specified in ERC-165.
     */
    function _supportsERC165Interface(address account, bytes4 interfaceId) private view
returns (bool) {
        // success determines whether the staticcall succeeded and result determines
        // whether the contract at account indicates support of _interfaceId
```

```
        (bool success, bool result) = _callERC165SupportsInterface(account, interfaceId);

        return (success && result);
    }

    /**
     * @notice Calls the function with selector 0x01ffc9a7 (ERC165) and suppresses throw
     * @param account The address of the contract to query for support of an interface
     * @param interfaceId The interface identifier, as specified in ERC-165
     * @return success true if the STATICCALL succeeded, false otherwise
     * @return result true if the STATICCALL succeeded and the contract at account
     * indicates support of the interface with identifier interfaceId, false otherwise
     */
    function _callERC165SupportsInterface(address account, bytes4 interfaceId)
        private
        view
        returns (bool success, bool result)
    {
        bytes memory encodedParams = abi.encodeWithSelector(_INTERFACE_ID_ERC165,
interfaceId);

        // solhint-disable-next-line no-inline-assembly
        assembly {
            let encodedParams_data := add(0x20, encodedParams)
            let encodedParams_size := mload(encodedParams)

            let output := mload(0x40)    // Find empty storage location using "free memory pointer"
            mstore(output, 0x0)

            success := staticcall(
                30000,                  // 30k gas
                account,                // To addr
                encodedParams_data,
                encodedParams_size,
                output,
                0x20                    // Outputs are 32 bytes long
            )

            result := mload(output)      // Load the result
        }
    }
}

// File: @openzeppelin/contracts/GSN/Context.sol

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
```

```
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://
github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
```

```
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns
(bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: @openzeppelin/contracts/math/SafeMath.sol

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
```

```
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
```

```
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message
on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256)
{
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
```

```
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20.sol

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20Mintable}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
```

```
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
```

```
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for `sender`'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public returns
(bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
transfer amount exceeds allowance"));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
```

```
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool)
{
        _approve(_msgSender(), spender, _allowances[_msgSender()]
[spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds
balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");
```

```
        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

     /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds
balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
```

```
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount,
"ERC20: burn amount exceeds allowance"));
    }
}

// File: @openzeppelin/contracts/introspection/IERC165.sol

/**
 * @dev Interface of the ERC165 standard, as defined in the
 * https://eips.ethereum.org/EIPS/eip-165[EIP].
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others ({ERC165Checker}).
 *
 * For an implementation, see {ERC165}.
 */
interface IERC165 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP section]
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}

// File: @openzeppelin/contracts/introspection/ERC165.sol

/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts may inherit from this and call {_registerInterface} to declare
 * their support of an interface.
 */
contract ERC165 is IERC165 {
    /*
     * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
     */
    bytes4 private constant _INTERFACE_ID_ERC165 = 0x01ffc9a7;

    /**
     * @dev Mapping of interface ids to whether or not it's supported.
     */
    mapping(bytes4 => bool) private _supportedInterfaces;

    constructor () internal {
        // Derived contracts need only register support for their own interfaces,
        // we register support for ERC165 itself here
        _registerInterface(_INTERFACE_ID_ERC165);
    }

    /**
```

```
     * @dev See {IERC165-supportsInterface}.
     *
     * Time complexity O(1), guaranteed to always use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool) {
        return _supportedInterfaces[interfaceId];
    }

    /**
     * @dev Registers the contract as an implementer of the interface defined by
     * `interfaceId`. Support of the actual ERC165 interface is automatic and
     * registering its interface id is not required.
     *
     * See {IERC165-supportsInterface}.
     *
     * Requirements:
     *
     * - `interfaceId` cannot be the ERC165 invalid interface (`0xffffffff`).
     */
    function _registerInterface(bytes4 interfaceId) internal {
        require(interfaceId != 0xffffffff, "ERC165: invalid interface id");
        _supportedInterfaces[interfaceId] = true;
    }
}

// File: erc-payable-token/contracts/token/ERC1363/IERC1363.sol

/**
 * @title IERC1363 Interface
 * @author Vittorio Minacori (https://github.com/vittominacori)
 * @dev Interface for a Payable Token contract as defined in
 *  https://github.com/ethereum/EIPs/issues/1363
 */
contract IERC1363 is IERC20, ERC165 {
    /*
     * Note: the ERC-165 identifier for this interface is 0x4bbee2df.
     * 0x4bbee2df ===
     *   bytes4(keccak256('transferAndCall(address,uint256)')) ^
     *   bytes4(keccak256('transferAndCall(address,uint256,bytes)')) ^
     *   bytes4(keccak256('transferFromAndCall(address,address,uint256)')) ^
     *   bytes4(keccak256('transferFromAndCall(address,address,uint256,bytes)'))
     */

    /*
     * Note: the ERC-165 identifier for this interface is 0xfb9ec8ce.
     * 0xfb9ec8ce ===
     *   bytes4(keccak256('approveAndCall(address,uint256)')) ^
     *   bytes4(keccak256('approveAndCall(address,uint256,bytes)'))
     */

    /**
     * @notice Transfer tokens from `msg.sender` to another address and then call
`onTransferReceived` on receiver
     * @param to address The address which you want to transfer to
```

```
     * @param value uint256 The amount of tokens to be transferred
     * @return true unless throwing
     */
    function transferAndCall(address to, uint256 value) public returns (bool);

    /**
     * @notice Transfer tokens from `msg.sender` to another address and then call
`onTransferReceived` on receiver
     * @param to address The address which you want to transfer to
     * @param value uint256 The amount of tokens to be transferred
     * @param data bytes Additional data with no specified format, sent in call to `to`
     * @return true unless throwing
     */
    function transferAndCall(address to, uint256 value, bytes memory data) public returns (bool);

    /**
     * @notice Transfer tokens from one address to another and then call `onTransferReceived`
on receiver
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 The amount of tokens to be transferred
     * @return true unless throwing
     */
    function transferFromAndCall(address from, address to, uint256 value) public returns (bool);


    /**
     * @notice Transfer tokens from one address to another and then call `onTransferReceived`
on receiver
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 The amount of tokens to be transferred
     * @param data bytes Additional data with no specified format, sent in call to `to`
     * @return true unless throwing
     */
    function transferFromAndCall(address from, address to, uint256 value, bytes memory data)
public returns (bool);

    /**
     * @notice Approve the passed address to spend the specified amount of tokens on behalf
of msg.sender
     * and then call `onApprovalReceived` on spender.
     * Beware that changing an allowance with this method brings the risk that someone may
use both the old
     * and the new allowance by unfortunate transaction ordering. One possible solution to
mitigate this
     * race condition is to first reduce the spender's allowance to 0 and set the desired value
afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     * @param spender address The address which will spend the funds
     * @param value uint256 The amount of tokens to be spent
     */
    function approveAndCall(address spender, uint256 value) public returns (bool);
```

```
    /**
     * @notice Approve the passed address to spend the specified amount of tokens on behalf
of msg.sender
     * and then call `onApprovalReceived` on spender.
     * Beware that changing an allowance with this method brings the risk that someone may
use both the old
     * and the new allowance by unfortunate transaction ordering. One possible solution to
mitigate this
     * race condition is to first reduce the spender's allowance to 0 and set the desired value
afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     * @param spender address The address which will spend the funds
     * @param value uint256 The amount of tokens to be spent
     * @param data bytes Additional data with no specified format, sent in call to `spender`
     */
    function approveAndCall(address spender, uint256 value, bytes memory data) public returns
(bool);
}

// File: erc-payable-token/contracts/token/ERC1363/IERC1363Receiver.sol

/**
 * @title IERC1363Receiver Interface
 * @author Vittorio Minacori (https://github.com/vittominacori)
 * @dev Interface for any contract that wants to support transferAndCall or
transferFromAndCall
 *  from ERC1363 token contracts as defined in
 *  https://github.com/ethereum/EIPs/issues/1363
 */
contract IERC1363Receiver {
    /*
     * Note: the ERC-165 identifier for this interface is 0x88a7ca5c.
     * 0x88a7ca5c ===
bytes4(keccak256("onTransferReceived(address,address,uint256,bytes)"))
     */

    /**
     * @notice Handle the receipt of ERC1363 tokens
     * @dev Any ERC1363 smart contract calls this function on the recipient
     * after a `transfer` or a `transferFrom`. This function MAY throw to revert and reject the
     * transfer. Return of other than the magic value MUST result in the
     * transaction being reverted.
     * Note: the token contract address is always the message sender.
     * @param operator address The address which called `transferAndCall` or
`transferFromAndCall` function
     * @param from address The address which are token transferred from
     * @param value uint256 The amount of tokens transferred
     * @param data bytes Additional data with no specified format
     * @return `bytes4(keccak256("onTransferReceived(address,address,uint256,bytes)"))`
     *  unless throwing
     */
    function onTransferReceived(address operator, address from, uint256 value, bytes memory
data) public returns (bytes4); // solhint-disable-line  max-line-length
}
```

```
// File: erc-payable-token/contracts/token/ERC1363/IERC1363Spender.sol

/**
 * @title IERC1363Spender Interface
 * @author Vittorio Minacori (https://github.com/vittominacori)
 * @dev Interface for any contract that wants to support approveAndCall
 *  from ERC1363 token contracts as defined in
 *  https://github.com/ethereum/EIPs/issues/1363
 */
contract IERC1363Spender {
    /*
     * Note: the ERC-165 identifier for this interface is 0x7b04a2d0.
     * 0x7b04a2d0 === bytes4(keccak256("onApprovalReceived(address,uint256,bytes)"))
     */

    /**
     * @notice Handle the approval of ERC1363 tokens
     * @dev Any ERC1363 smart contract calls this function on the recipient
     * after an `approve`. This function MAY throw to revert and reject the
     * approval. Return of other than the magic value MUST result in the
     * transaction being reverted.
     * Note: the token contract address is always the message sender.
     * @param owner address The address which called `approveAndCall` function
     * @param value uint256 The amount of tokens to be spent
     * @param data bytes Additional data with no specified format
     * @return `bytes4(keccak256("onApprovalReceived(address,uint256,bytes)"))`
     *  unless throwing
     */
    function onApprovalReceived(address owner, uint256 value, bytes memory data) public
returns (bytes4);
}

// File: erc-payable-token/contracts/token/ERC1363/ERC1363.sol

/**
 * @title ERC1363
 * @author Vittorio Minacori (https://github.com/vittominacori)
 * @dev Implementation of an ERC1363 interface
 */
contract ERC1363 is ERC20, IERC1363 {
    using Address for address;

    /*
     * Note: the ERC-165 identifier for this interface is 0x4bbee2df.
     * 0x4bbee2df ===
     *   bytes4(keccak256('transferAndCall(address,uint256)')) ^
     *   bytes4(keccak256('transferAndCall(address,uint256,bytes)')) ^
     *   bytes4(keccak256('transferFromAndCall(address,address,uint256)')) ^
     *   bytes4(keccak256('transferFromAndCall(address,address,uint256,bytes)'))
     */
    bytes4 internal constant _INTERFACE_ID_ERC1363_TRANSFER = 0x4bbee2df;

    /*
```

```
     * Note: the ERC-165 identifier for this interface is 0xfb9ec8ce.
     * 0xfb9ec8ce ===
     *   bytes4(keccak256('approveAndCall(address,uint256)')) ^
     *   bytes4(keccak256('approveAndCall(address,uint256,bytes)'))
     */
    bytes4 internal constant _INTERFACE_ID_ERC1363_APPROVE = 0xfb9ec8ce;

    // Equals to `bytes4(keccak256("onTransferReceived(address,address,uint256,bytes)"))`
    // which can be also obtained as `IERC1363Receiver(0).onTransferReceived.selector`
    bytes4 private constant _ERC1363_RECEIVED = 0x88a7ca5c;

    // Equals to `bytes4(keccak256("onApprovalReceived(address,uint256,bytes)"))`
    // which can be also obtained as `IERC1363Spender(0).onApprovalReceived.selector`
    bytes4 private constant _ERC1363_APPROVED = 0x7b04a2d0;

    constructor() public {
        // register the supported interfaces to conform to ERC1363 via ERC165
        _registerInterface(_INTERFACE_ID_ERC1363_TRANSFER);
        _registerInterface(_INTERFACE_ID_ERC1363_APPROVE);
    }

    function transferAndCall(address to, uint256 value) public returns (bool) {
        return transferAndCall(to, value, "");
    }

    function transferAndCall(address to, uint256 value, bytes memory data) public returns (bool)
{
        require(transfer(to, value));
        require(_checkAndCallTransfer(msg.sender, to, value, data));
        return true;
    }

    function transferFromAndCall(address from, address to, uint256 value) public returns (bool) {
        return transferFromAndCall(from, to, value, "");
    }

    function transferFromAndCall(address from, address to, uint256 value, bytes memory data)
public returns (bool) {
        require(transferFrom(from, to, value));
        require(_checkAndCallTransfer(from, to, value, data));
        return true;
    }

    function approveAndCall(address spender, uint256 value) public returns (bool) {
        return approveAndCall(spender, value, "");
    }

    function approveAndCall(address spender, uint256 value, bytes memory data) public returns
(bool) {
        approve(spender, value);
        require(_checkAndCallApprove(spender, value, data));
        return true;
    }
```

```
    /**
     * @dev Internal function to invoke `onTransferReceived` on a target address
     *  The call is not executed if the target address is not a contract
     * @param from address Representing the previous owner of the given token value
     * @param to address Target address that will receive the tokens
     * @param value uint256 The amount mount of tokens to be transferred
     * @param data bytes Optional data to send along with the call
     * @return whether the call correctly returned the expected magic value
     */
    function _checkAndCallTransfer(address from, address to, uint256 value, bytes memory
data) internal returns (bool) {
        if (!to.isContract()) {
            return false;
        }
        bytes4 retval = IERC1363Receiver(to).onTransferReceived(
            msg.sender, from, value, data
        );
        return (retval == _ERC1363_RECEIVED);
    }

    /**
     * @dev Internal function to invoke `onApprovalReceived` on a target address
     *  The call is not executed if the target address is not a contract
     * @param spender address The address which will spend the funds
     * @param value uint256 The amount of tokens to be spent
     * @param data bytes Optional data to send along with the call
     * @return whether the call correctly returned the expected magic value
     */
    function _checkAndCallApprove(address spender, uint256 value, bytes memory data)
internal returns (bool) {
        if (!spender.isContract()) {
            return false;
        }
        bytes4 retval = IERC1363Spender(spender).onApprovalReceived(
            msg.sender, value, data
        );
        return (retval == _ERC1363_APPROVED);
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Detailed.sol

/**
 * @dev Optional functions from the ERC20 standard.
 */
contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of
     * these values are immutable: they can only be set once during
     * construction.
```

```
     */
    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }

    /**
     * @dev Returns the name of the token.
     */
    function name() public view returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

// File: @openzeppelin/contracts/access/Roles.sol

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev Give an account access to this role.
     */
    function add(Role storage role, address account) internal {
```

```
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
}

// File: @openzeppelin/contracts/access/roles/MinterRole.sol

contract MinterRole is Context {
    using Roles for Roles.Role;

    event MinterAdded(address indexed account);
    event MinterRemoved(address indexed account);

    Roles.Role private _minters;

    constructor () internal {
        _addMinter(_msgSender());
    }

    modifier onlyMinter() {
        require(isMinter(_msgSender()), "MinterRole: caller does not have the Minter role");
        _;
    }

    function isMinter(address account) public view returns (bool) {
        return _minters.has(account);
    }

    function addMinter(address account) public onlyMinter {
        _addMinter(account);
    }

    function renounceMinter() public {
        _removeMinter(_msgSender());
    }

    function _addMinter(address account) internal {
```

```
        _minters.add(account);
        emit MinterAdded(account);
    }

    function _removeMinter(address account) internal {
        _minters.remove(account);
        emit MinterRemoved(account);
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Mintable.sol

/**
 * @dev Extension of {ERC20} that adds a set of accounts with the {MinterRole},
 * which have permission to mint (create) new tokens as they see fit.
 *
 * At construction, the deployer of the contract is the only minter.
 */
contract ERC20Mintable is ERC20, MinterRole {
    /**
     * @dev See {ERC20-_mint}.
     *
     * Requirements:
     *
     * - the caller must have the {MinterRole}.
     */
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount);
        return true;
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Capped.sol

/**
 * @dev Extension of {ERC20Mintable} that adds a cap to the supply of tokens.
 */
contract ERC20Capped is ERC20Mintable {
    uint256 private _cap;

    /**
     * @dev Sets the value of the `cap`. This value is immutable, it can only be
     * set once during construction.
     */
    constructor (uint256 cap) public {
        require(cap > 0, "ERC20Capped: cap is 0");
        _cap = cap;
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view returns (uint256) {
        return _cap;
```

```
    }

    /**
     * @dev See {ERC20Mintable-mint}.
     *
     * Requirements:
     *
     * - `value` must not cause the total supply to go over the cap.
     */
    function _mint(address account, uint256 value) internal {
        require(totalSupply().add(value) <= _cap, "ERC20Capped: cap exceeded");
        super._mint(account, value);
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20Burnable.sol

/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev See {ERC20-_burnFrom}.
     */
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner;
```

```
event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

/**
 * @dev Initializes the contract setting the deployer as the initial owner.
 */
constructor () internal {
    _owner = _msgSender();
    emit OwnershipTransferred(address(0), _owner);
}

/**
 * @dev Returns the address of the current owner.
 */
function owner() public view returns (address) {
    return _owner;
}

/**
 * @dev Throws if called by any account other than the owner.
 */
modifier onlyOwner() {
    require(isOwner(), "Ownable: caller is not the owner");
    _;
}

/**
 * @dev Returns true if the caller is the current owner.
 */
function isOwner() public view returns (bool) {
    return _msgSender() == _owner;
}

/**
 * @dev Leaves the contract without owner. It will not be possible to call
 * `onlyOwner` functions anymore. Can only be called by the current owner.
 *
 * NOTE: Renouncing ownership will leave the contract without an owner,
 * thereby removing any functionality that is only available to the owner.
 */
function renounceOwnership() public onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address newOwner) public onlyOwner {
    _transferOwnership(newOwner);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
```

```
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: eth-token-recover/contracts/TokenRecover.sol

/**
 * @title TokenRecover
 * @author Vittorio Minacori (https://github.com/vittominacori)
 * @dev Allow to recover any ERC20 sent into the contract for error
 */
contract TokenRecover is Ownable {

    /**
     * @dev Remember that only owner can call so be careful when use on contracts generated
from other contracts.
     * @param tokenAddress The token contract address
     * @param tokenAmount Number of tokens to be sent
     */
    function recoverERC20(address tokenAddress, uint256 tokenAmount) public onlyOwner {
        IERC20(tokenAddress).transfer(owner(), tokenAmount);
    }
}

// File: ico-maker/contracts/access/roles/OperatorRole.sol

contract OperatorRole {
    using Roles for Roles.Role;

    event OperatorAdded(address indexed account);
    event OperatorRemoved(address indexed account);

    Roles.Role private _operators;

    constructor() internal {
        _addOperator(msg.sender);
    }

    modifier onlyOperator() {
        require(isOperator(msg.sender));
        _;
    }

    function isOperator(address account) public view returns (bool) {
        return _operators.has(account);
    }

    function addOperator(address account) public onlyOperator {
        _addOperator(account);
    }
```

```
    function renounceOperator() public {
        _removeOperator(msg.sender);
    }

    function _addOperator(address account) internal {
        _operators.add(account);
        emit OperatorAdded(account);
    }

    function _removeOperator(address account) internal {
        _operators.remove(account);
        emit OperatorRemoved(account);
    }
}

// File: ico-maker/contracts/token/ERC20/BaseERC20Token.sol

/**
 * @title BaseERC20Token
 * @author Vittorio Minacori (https://github.com/vittominacori)
 * @dev Implementation of the BaseERC20Token
 */
contract BaseERC20Token is ERC20Detailed, ERC20Capped, ERC20Burnable, OperatorRole,
TokenRecover {

    event MintFinished();
    event TransferEnabled();

    // indicates if minting is finished
    bool private _mintingFinished = false;

    // indicates if transfer is enabled
    bool private _transferEnabled = false;

    /**
     * @dev Tokens can be minted only before minting finished.
     */
    modifier canMint() {
        require(!_mintingFinished);
        _;
    }

    /**
     * @dev Tokens can be moved only after if transfer enabled or if you are an approved
operator.
     */
    modifier canTransfer(address from) {
        require(_transferEnabled || isOperator(from));
        _;
    }

    /**
     * @param name Name of the token
```

```
     * @param symbol A symbol to be used as ticker
     * @param decimals Number of decimals. All the operations are done using the smallest and
indivisible token unit
     * @param cap Maximum number of tokens mintable
     * @param initialSupply Initial token supply
     */
    constructor(
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 cap,
        uint256 initialSupply
    )
        public
        ERC20Detailed(name, symbol, decimals)
        ERC20Capped(cap)
    {
        if (initialSupply > 0) {
            _mint(owner(), initialSupply);
        }
    }

    /**
     * @return if minting is finished or not.
     */
    function mintingFinished() public view returns (bool) {
        return _mintingFinished;
    }

    /**
     * @return if transfer is enabled or not.
     */
    function transferEnabled() public view returns (bool) {
        return _transferEnabled;
    }

    /**
     * @dev Function to mint tokens
     * @param to The address that will receive the minted tokens.
     * @param value The amount of tokens to mint.
     * @return A boolean that indicates if the operation was successful.
     */
    function mint(address to, uint256 value) public canMint returns (bool) {
        return super.mint(to, value);
    }

    /**
     * @dev Transfer token to a specified address
     * @param to The address to transfer to.
     * @param value The amount to be transferred.
     * @return A boolean that indicates if the operation was successful.
     */
    function transfer(address to, uint256 value) public canTransfer(msg.sender) returns (bool) {
        return super.transfer(to, value);
```

```
    }

    /**
     * @dev Transfer tokens from one address to another.
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 the amount of tokens to be transferred
     * @return A boolean that indicates if the operation was successful.
     */
    function transferFrom(address from, address to, uint256 value) public canTransfer(from)
returns (bool) {
        return super.transferFrom(from, to, value);
    }

    /**
     * @dev Function to stop minting new tokens.
     */
    function finishMinting() public onlyOwner canMint {
        _mintingFinished = true;

        emit MintFinished();
    }

    /**
   * @dev Function to enable transfers.
   */
    function enableTransfer() public onlyOwner {
        _transferEnabled = true;

        emit TransferEnabled();
    }

    /**
     * @dev remove the `operator` role from address
     * @param account Address you want to remove role
     */
    function removeOperator(address account) public onlyOwner {
        _removeOperator(account);
    }

    /**
     * @dev remove the `minter` role from address
     * @param account Address you want to remove role
     */
    function removeMinter(address account) public onlyOwner {
        _removeMinter(account);
    }
}

// File: ico-maker/contracts/token/ERC1363/BaseERC1363Token.sol

/**
 * @title BaseERC1363Token
 * @author Vittorio Minacori (https://github.com/vittominacori)
```

```
 * @dev Implementation of the BaseERC20Token with ERC1363 behaviours
 */
contract BaseERC1363Token is BaseERC20Token, ERC1363 {

    /**
     * @param name Name of the token
     * @param symbol A symbol to be used as ticker
     * @param decimals Number of decimals. All the operations are done using the smallest and
indivisible token unit
     * @param cap Maximum number of tokens mintable
     * @param initialSupply Initial token supply
     */
    constructor(
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 cap,
        uint256 initialSupply
    )
        public
        BaseERC20Token(name, symbol, decimals, cap, initialSupply)
    {} // solhint-disable-line no-empty-blocks
}

// File: contracts/ERC20Token.sol

/**
 * @title ERC20Token
 * @author Vittorio Minacori (https://github.com/vittominacori)
 * @dev Implementation of a BaseERC1363Token
 */
contract ERC20Token is BaseERC1363Token {

    string public builtOn = "https://vittominacori.github.io/erc20-generator";

    constructor(
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 cap,
        uint256 initialSupply,
        bool transferEnabled
    )
        public
        BaseERC1363Token(name, symbol, decimals, cap, initialSupply)
    {
        if (transferEnabled) {
            enableTransfer();
        }
    }
}
```