/**

*Submitted for verification at BscScan.com on 2024-09-09

*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

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

// OpenZeppelin Contracts (last updated v5.0.0) (token/ERC20/IERC20.sol)

```
pragma solidity ^0.8.20;


/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @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);

    /**
     * @dev Returns the value of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the value of tokens owned by `account`.
     */
```

```
    function balanceOf(address
account) external view returns
(uint256);


    /**
     * @dev Moves a `value` amount
of tokens from the caller's
account to `to`.
     *
     * Returns a boolean value
indicating whether the operation
succeeded.
     *
     * Emits a {Transfer} event.
     */
```

```
    function transfer(address to, uint256 value) 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 a `value` amount of tokens 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 value) external
returns (bool);

    /**
    * @dev Moves a `value` amount
of tokens from `from` to `to` using
the
```

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

// File: @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol

// OpenZeppelin Contracts (last updated v5.0.0) (token/ERC20/extensions/IERC20Metadata.sol)

pragma solidity ^0.8.20;

```
/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);
```

```
    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}
```

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

// OpenZeppelin Contracts (last updated v5.0.1) (utils/Context.sol)

pragma solidity ^0.8.20;

/**
 * @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 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.
 */
abstract contract Context {
    function _msgSender() internal
view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal
view virtual returns (bytes
calldata) {
        return msg.data;
```

```
    }


    function _contextSuffixLength()
internal view virtual returns
(uint256) {
        return 0;
    }
}


// File:
@openzeppelin/contracts/interfa
ces/draft-IERC6093.sol
```

```
// OpenZeppelin Contracts (last updated v5.0.0) (interfaces/draft-IERC6093.sol)

pragma solidity ^0.8.20;


/**
 * @dev Standard ERC20 Errors
 * Interface of the https://eips.ethereum.org/EIPS/eip-6093[ERC-6093] custom errors for ERC20 tokens.
 */
interface IERC20Errors {
    /**
```

```
    * @dev Indicates an error related to the current `balance` of a `sender`. Used in transfers.

    * @param sender Address whose tokens are being transferred.

    * @param balance Current balance for the interacting account.

    * @param needed Minimum amount required to perform a transfer.

    */

   error ERC20InsufficientBalance(addres
```

```
s sender, uint256 balance, uint256
needed);


    /**
     * @dev Indicates a failure with
the token `sender`. Used in
transfers.
     * @param sender Address
whose tokens are being
transferred.
     */
    error
ERC20InvalidSender(address
sender);
```

```
    /**
     * @dev Indicates a failure with the token `receiver`. Used in transfers.
     * @param receiver Address to which tokens are being transferred.
     */
    error ERC20InvalidReceiver(address receiver);

    /**
     * @dev Indicates a failure with the `spender`’s `allowance`. Used
```

in transfers.

* @param spender Address that may be allowed to operate on tokens without being their owner.

* @param allowance Amount of tokens a `spender` is allowed to operate with.

* @param needed Minimum amount required to perform a transfer.

*/

error ERC20InsufficientAllowance(address spender, uint256 allowance, uint256 needed);

```
    /**
     * @dev Indicates a failure with the `approver` of a token to be approved. Used in approvals.
     * @param approver Address initiating an approval operation.
     */
    error ERC20InvalidApprover(address approver);

    /**
     * @dev Indicates a failure with the `spender` to be approved.
```

```
Used in approvals.
     * @param spender Address that
may be allowed to operate on
tokens without being their owner.
     */
    error
ERC20InvalidSpender(address
spender);
}


/**
 * @dev Standard ERC721 Errors
 * Interface of the
https://eips.ethereum.org/EIPS/ei
```

p-6093[ERC-6093] custom errors for ERC721 tokens.

*/

interface IERC721Errors {

/**

* @dev Indicates that an address can't be an owner. For example, `address(0)` is a forbidden owner in EIP-20.

* Used in balance queries.

* @param owner Address of the current owner of a token.

*/

error ERC721InvalidOwner(address

```
owner);

    /**
     * @dev Indicates a `tokenId` whose `owner` is the zero address.
     * @param tokenId Identifier number of a token.
     */
    error ERC721NonexistentToken(uint256 tokenId);

    /**
     * @dev Indicates an error related to the ownership over a
```

particular token. Used in transfers.

* @param sender Address whose tokens are being transferred.

* @param tokenId Identifier number of a token.

* @param owner Address of the current owner of a token.

*/

error ERC721IncorrectOwner(address sender, uint256 tokenId, address owner);

```
    /**
     * @dev Indicates a failure with the token `sender`. Used in transfers.
     * @param sender Address whose tokens are being transferred.
     */
    error ERC721InvalidSender(address sender);

    /**
     * @dev Indicates a failure with the token `receiver`. Used in
```

transfers.

* @param receiver Address to which tokens are being transferred.

*/

error ERC721InvalidReceiver(address receiver);

/**

* @dev Indicates a failure with the `operator`’s approval. Used in transfers.

* @param operator Address that may be allowed to operate on

tokens without being their owner.

* @param tokenId Identifier number of a token.

*/

error ERC721InsufficientApproval(address operator, uint256 tokenId);

/**

* @dev Indicates a failure with the `approver` of a token to be approved. Used in approvals.

* @param approver Address initiating an approval operation.

*/

```
    error
ERC721InvalidApprover(address
approver);


    /**
     * @dev Indicates a failure with
the `operator` to be approved.
Used in approvals.
     * @param operator Address
that may be allowed to operate on
tokens without being their owner.
     */
    error
ERC721InvalidOperator(address
operator);
```

```
}


/**
 * @dev Standard ERC1155 Errors
 * Interface of the https://eips.ethereum.org/EIPS/eip-6093[ERC-6093] custom errors for ERC1155 tokens.
 */
interface IERC1155Errors {
    /**
     * @dev Indicates an error related to the current `balance` of a `sender`. Used in transfers.
```

```
     * @param sender Address
whose tokens are being
transferred.

     * @param balance Current
balance for the interacting
account.

     * @param needed Minimum
amount required to perform a
transfer.

     * @param tokenId Identifier
number of a token.

     */

    error
ERC1155InsufficientBalance(addr
```

```
ess sender, uint256 balance,
uint256 needed, uint256 tokenId);


    /**
     * @dev Indicates a failure with
the token `sender`. Used in
transfers.
     * @param sender Address
whose tokens are being
transferred.
     */
    error
ERC1155InvalidSender(address
sender);
```

```
    /**
     * @dev Indicates a failure with the token `receiver`. Used in transfers.
     * @param receiver Address to which tokens are being transferred.
     */
    error ERC1155InvalidReceiver(address receiver);

    /**
     * @dev Indicates a failure with the `operator`’s approval. Used in
```

transfers.

* @param operator Address that may be allowed to operate on tokens without being their owner.

* @param owner Address of the current owner of a token.

*/

error ERC1155MissingApprovalForAll(address operator, address owner);

/**

* @dev Indicates a failure with the `approver` of a token to be approved. Used in approvals.

```
     * @param approver Address initiating an approval operation.
     */
    error ERC1155InvalidApprover(address approver);

    /**
     * @dev Indicates a failure with the `operator` to be approved. Used in approvals.
     * @param operator Address that may be allowed to operate on tokens without being their owner.
     */
```

```
    error
ERC1155InvalidOperator(address
operator);


    /**
     * @dev Indicates an array
length mismatch between ids and
values in a safeBatchTransferFrom
operation.
     * Used in batch transfers.
     * @param idsLength Length of
the array of token identifiers
     * @param valuesLength Length
of the array of token amounts
     */
```

error ERC1155InvalidArrayLength(uint256 idsLength, uint256 valuesLength);
}

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

// OpenZeppelin Contracts (last updated v5.0.0) (token/ERC20/ERC20.sol)

```
pragma solidity ^0.8.20;

/**
 * @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}.
*
* TIP: For a detailed writeup see our guide
*
https://forum.openzeppelin.com/t/how-to-implement-erc20-supply-mechanisms/226[How
* to implement supply mechanisms].
*
* The default value of {decimals} is 18. To change this, you should

override

* this function so it returns a different value.

*

* We have followed general OpenZeppelin Contracts guidelines: functions revert

* instead 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.
 */
abstract contract ERC20 is
Context, IERC20,
IERC20Metadata, IERC20Errors {
```

```
    mapping(address account =>
uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
```

```
    * @dev Sets the values for
{name} and {symbol}.
    *
    * All two of these values are
immutable: they can only be set
once during
    * construction.
    */
   constructor(string memory
name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
   }
```

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

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
```

```
    */
    function symbol() public view
virtual 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. This is the default value returned by this function, unless

* it's overridden.

*

* 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 virtual returns (uint8) {
        return 18;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view virtual returns (uint256) {
```

```
        return _totalSupply;
    }

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

    /**
```

```
    * @dev See {IERC20-transfer}.
    *
    * Requirements:
    *
    * - `to` cannot be the zero address.
    * - the caller must have a balance of at least `value`.
    */
   function transfer(address to, uint256 value) public virtual returns (bool) {
       address owner = _msgSender();
```

```
        _transfer(owner, to, value);
        return true;
    }

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

```
    /**
     * @dev See {IERC20-approve}.
     *
     * NOTE: If `value` is the maximum `uint256`, the allowance is not updated on
     * `transferFrom`. This is semantically equivalent to an infinite approval.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
```

```
     */
    function approve(address
spender, uint256 value) public
virtual returns (bool) {
        address owner =
_msgSender();
        _approve(owner, spender,
value);
        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}.

*

* NOTE: Does not update the allowance if the current allowance

* is the maximum `uint256`.

*

* Requirements:

*

* - `from` and `to` cannot be the

```
zero address.
     * - `from` must have a balance of at least `value`.
     * - the caller must have allowance for ``from``'s tokens of at least
     * `value`.
     */
    function transferFrom(address from, address to, uint256 value) public virtual returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, value);
```

```
        _transfer(from, to, value);
        return true;
    }

    /**
     * @dev Moves a `value` amount of tokens from `from` to `to`.
     *
     * This internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
```

```
    *
    * Emits a {Transfer} event.
    *
    * NOTE: This function is not virtual, {_update} should be overridden instead.
    */
    function _transfer(address from, address to, uint256 value) internal {
        if (from == address(0)) {
            revert ERC20InvalidSender(address(0));
        }
```

```
        if (to == address(0)) {
            revert
ERC20InvalidReceiver(address(0))
;
        }
        _update(from, to, value);
    }


    /**
     * @dev Transfers a `value`
amount of tokens from `from` to
`to`, or alternatively mints (or
burns) if `from`
     * (or `to`) is the zero address. All
customizations to transfers,
```

```
mints, and burns should be done
by overriding
    * this function.
    *
    * Emits a {Transfer} event.
    */
   function _update(address from,
address to, uint256 value) internal
virtual {
        if (from == address(0)) {
            // Overflow check required:
The rest of the code assumes that
totalSupply never overflows
            _totalSupply += value;
        } else {
```

```
        uint256 fromBalance =
_balances[from];
        if (fromBalance < value) {
            revert
ERC20InsufficientBalance(from,
fromBalance, value);
        }
        unchecked {
            // Overflow not possible:
value <= fromBalance <=
totalSupply.
            _balances[from] =
fromBalance - value;
        }
    }
```

```
        if (to == address(0)) {
            unchecked {
                // Overflow not possible: value <= totalSupply or value <= fromBalance <= totalSupply.
                _totalSupply -= value;
            }
        } else {
            unchecked {
                // Overflow not possible: balance + value is at most totalSupply, which we know fits into a uint256.
```

```
            _balances[to] += value;
        }
    }

    emit Transfer(from, to, value);
}

/**
 * @dev Creates a `value` amount of tokens and assigns them to `account`, by transferring it from address(0).
 * Relies on the `_update` mechanism
```

```
    *
    * Emits a {Transfer} event with `from` set to the zero address.
    *
    * NOTE: This function is not virtual, {_update} should be overridden instead.
    */
   function _mint(address account, uint256 value) internal {
        if (account == address(0)) {
            revert ERC20InvalidReceiver(address(0));
        }
```

```
        _update(address(0), account, value);
    }

    /**
     * @dev Destroys a `value` amount of tokens from `account`, lowering the total supply.
     * Relies on the `_update` mechanism.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
```

```
     * NOTE: This function is not
virtual, {_update} should be
overridden instead
     */
    function _burn(address account,
uint256 value) internal {
        if (account == address(0)) {
            revert
ERC20InvalidSender(address(0));
        }
        _update(account, address(0),
value);
    }
```

```
    /**
     * @dev Sets `value` as the allowance of `spender` over the `owner` s tokens.
     *
     * This 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.
    *
    * Overrides to this logic should be done to the variant with an additional `bool emitEvent` argument.
    */
    function _approve(address owner, address spender, uint256 value) internal {
```

```
        _approve(owner, spender,
value, true);
    }


    /**
     * @dev Variant of {_approve}
with an optional flag to enable or
disable the {Approval} event.
     *
     * By default (when calling
{_approve}) the flag is set to true.
On the other hand, approval
changes made by
     * `_spendAllowance` during the
`transferFrom` operation set the
```

flag to false. This saves gas by not emitting any

* `Approval` event during `transferFrom` operations.

*

* Anyone who wishes to continue emitting `Approval` events on the`transferFrom` operation can force the flag to

* true using the following override:

* ```

* function _approve(address owner, address spender, uint256

```
value, bool) internal virtual
override {
    *     super._approve(owner,
spender, value, true);
    * }
    * ```
    *
    * Requirements are the same as
{_approve}.
    */
   function _approve(address
owner, address spender, uint256
value, bool emitEvent) internal
virtual {
        if (owner == address(0)) {
```

```
            revert
ERC20InvalidApprover(address(0)
);
        }
        if (spender == address(0)) {
            revert
ERC20InvalidSpender(address(0))
;
        }
        _allowances[owner][spender]
= value;
        if (emitEvent) {
            emit Approval(owner,
spender, value);
        }
```

```
    }

    /**
     * @dev Updates `owner` s allowance for `spender` based on spent `value`.
     *
     * Does not update the allowance value in case of infinite allowance.
     * Revert if not enough allowance is available.
     *
     * Does not emit an {Approval} event.
```

```
    */
    function
_spendAllowance(address owner,
address spender, uint256 value)
internal virtual {
        uint256 currentAllowance =
allowance(owner, spender);
        if (currentAllowance !=
type(uint256).max) {
            if (currentAllowance < value)
{
                revert
ERC20InsufficientAllowance(spen
der, currentAllowance, value);
            }
```

```
            unchecked {
                _approve(owner, spender, currentAllowance - value, false);
            }
        }
    }
}

// File: @openzeppelin/contracts/access/Ownable.sol
```

```
// OpenZeppelin Contracts (last updated v5.0.0) (access/Ownable.sol)

pragma solidity ^0.8.20;


/**
 * @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.
```

*

* The initial owner is set to the address provided by the deployer. This can

* later be changed with {transferOwnership}.

*

* 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.

*/

```
abstract contract Ownable is
Context {
    address private _owner;


    /**
     * @dev The caller account is not
authorized to perform an
operation.
     */
    error
OwnableUnauthorizedAccount(ad
dress account);


    /**
```

```
    * @dev The owner is not a valid owner account. (eg. `address(0)`)
    */
    error OwnableInvalidOwner(address owner);

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

    /**
    * @dev Initializes the contract setting the address provided by
```

```
the deployer as the initial owner.
     */
    constructor(address
initialOwner) {
        if (initialOwner == address(0))
{
            revert
OwnableInvalidOwner(address(0))
;
        }

_transferOwnership(initialOwner);
    }

    /**
```

```
    * @dev Throws if called by any account other than the owner.
    */
    modifier onlyOwner() {
        _checkOwner();
        _;
    }

    /**
    * @dev Returns the address of the current owner.
    */
    function owner() public view virtual returns (address) {
```

```
        return _owner;
    }


    /**
     * @dev Throws if the sender is not the owner.
     */
    function _checkOwner() internal view virtual {
        if (owner() != _msgSender()) {
            revert OwnableUnauthorizedAccount(_msgSender());
        }
```

```
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby disabling any functionality that is only available
```

```
to the owner.
    */
    function renounceOwnership() public virtual onlyOwner {

_transferOwnership(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 virtual
onlyOwner {
        if (newOwner == address(0)) {
            revert
OwnableInvalidOwner(address(0))
;
        }

_transferOwnership(newOwner);
    }

    /**
```

```
    * @dev Transfers ownership of the contract to a new account (`newOwner`).
    * Internal function without access restriction.
    */
   function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
   }
```

}

// File: @openzeppelin/contracts/security/Pausable.sol

// OpenZeppelin Contracts (last updated v4.7.0) (security/Pausable.sol)

pragma solidity ^0.8.0;

```
/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
```

```
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
abstract contract Pausable is Context {
    /**
     * @dev Emitted when the pause is triggered by `account`.
     */
    event Paused(address account);
```

```
    /**
     * @dev Emitted when the pause is lifted by `account`.
     */
    event Unpaused(address account);

    bool private _paused;

    /**
     * @dev Initializes the contract in unpaused state.
     */
    constructor() {
```

```
        _paused = false;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
```

```
        _requireNotPaused();
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
```

```
    modifier whenPaused() {
        _requirePaused();
        _;
    }


    /**
     * @dev Returns true if the contract is paused, and false otherwise.
     */
    function paused() public view virtual returns (bool) {
        return _paused;
    }
```

```
    /**
     * @dev Throws if the contract is paused.
     */
    function _requireNotPaused() internal view virtual {
        require(!paused(), "Pausable: paused");
    }

    /**
     * @dev Throws if the contract is not paused.
```

```
    */
    function _requirePaused() internal view virtual {
        require(paused(), "Pausable: not paused");
    }

    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
```

```
    */
    function _pause() internal
virtual whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
    }

    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be
paused.
```

```
    */
    function _unpause() internal
virtual whenPaused {
        _paused = false;
        emit
Unpaused(_msgSender());
    }
}

// File: GBDCoin.sol


pragma solidity ^0.8.0;
```

```
contract GBDCoin is ERC20,
Ownable, Pausable {
    string private constant _name =
"Global Business Development";
    string private constant _symbol
= "GBD";
    uint8 private constant _decimals
= 18;
    uint256 private _totalSupply =
21_000_000 *
10**uint256(_decimals); //
Example initial supply
```

// Corrected addresses with proper checksum

address public constant WBTC = 0x58730ae0FAA10d73b0cDdb5e7b87C3594f7a20CB;

address public constant CLASSIC_USDC = 0x7DB54758503c25200B331131C219acD64CFa74c8;

address public constant BTCBR = 0x58730ae0FAA10d73b0cDdb5e7b87C3594f7a20CB;

address public constant BGEO = 0x58730ae0FAA10d73b0cDdb5e7

b87C3594f7a20CB;

address public constant BLK = 0xc0E6AD13BD58413Ed308729b688d601243E1CF77;

address public constant TRX = 0x9Dab87a99c6b5c516332894331A442Ea2890126C;

address public constant BABY_DOGE_ZILLA = 0x3749DB761943192C29E6931AC2F4ce8107Fe4C17;

mapping(address => uint256) public backingTokens;

```
    bool public transfersEnabled =
true;
    bool public swappingEnabled =
true;


    constructor(address
initialOwner) ERC20(_name,
_symbol) Ownable(initialOwner) {
        _mint(msg.sender,
_totalSupply);


        // Initialize backing tokens
with updated CLASSIC_USDC
amount
```

backingTokens[CLASSIC_USDC] = 100_000_000 * 10**18; // Updated Amount

backingTokens[WBTC] = 200_000 * 10**8; // Amount with decimals

backingTokens[BTCBR] = 1_000_000_000_000 * 10**18; // Amount with decimals

backingTokens[BGEO] = 20_000_000_000_000_000_000; // Amount with decimals

backingTokens[BLK] = 2_000_000_000 * 10**18; // Amount with decimals

```
        backingTokens[TRX] =
100_000_000_000 * 10**18; //
Amount with decimals

backingTokens[BABY_DOGE_ZILLA
] =
25_000_000_000_000_000_000
_000_000; // Amount with
decimals
    }

    // Functions

    function enableTransfers()
external onlyOwner {
```

```
        transfersEnabled = true;
    }

    function disableTransfers()
external onlyOwner {
        transfersEnabled = false;
    }

    function enableSwapping()
external onlyOwner {
        swappingEnabled = true;
    }
```

```
    function disableSwapping()
external onlyOwner {
        swappingEnabled = false;
    }


    function rebase(uint256
percentage) external onlyOwner {
        require(percentage > 0,
"Percentage must be positive");
        uint256 rebaseAmount =
(_totalSupply * percentage) / 100;
        _mint(msg.sender,
rebaseAmount);
        _totalSupply +=
rebaseAmount;
```

```
    }


    function
distributeRewards(address[]
calldata recipients, uint256
rewardAmount) external
onlyOwner {
        for (uint256 i = 0; i <
recipients.length; i++) {
            _transfer(msg.sender,
recipients[i], rewardAmount);
        }
    }
```

```
    function changeSupply(uint256
newSupply) external onlyOwner {
        _totalSupply = newSupply;
    }

    function addLiquidity(uint256
amount) external onlyOwner {
        // Placeholder for adding
liquidity
    }

    function
rewardSocialPromoters(address[]
calldata promoters, uint256
```

```
rewardAmount) external
onlyOwner {
        for (uint256 i = 0; i <
promoters.length; i++) {
            _transfer(msg.sender,
promoters[i], rewardAmount);
        }
    }

    function pause() external
onlyOwner {
        _pause();
    }
```

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


    function transfer(address
recipient, uint256 amount) public
override whenNotPaused returns
(bool) {
        require(transfersEnabled,
"Transfers are currently
disabled");
        return
super.transfer(recipient, amount);
    }
```

```
    function transferFrom(address sender, address recipient, uint256 amount) public override whenNotPaused returns (bool) {
        require(transfersEnabled, "Transfers are currently disabled");
        return super.transferFrom(sender, recipient, amount);
    }
}
```