AlphaXCoin (AXC) Technical Specification

Document Version: 1.0
Last Updated: May 18, 2025
Contract Version: 0.8.24
Status: Production

Table of Contents

---

1. Introduction

This technical specification document describes the implementation details, architecture, and functionality of the AlphaXCoinEnhanced (AXC) smart contract. The contract is designed to provide a comprehensive token ecosystem that includes multi-phase token sales, promotional incentives, staking mechanisms, referral rewards, and revenue sharing capabilities.

1.1 Purpose

The purpose of this document is to provide a detailed technical reference for developers, auditors, and technical stakeholders involved in the deployment, integration, or evaluation of the AlphaXCoin smart contract.

1.2 Scope

This document covers the technical aspects of the AlphaXCoin smart contract, including:

- Contract architecture and design patterns
- Function specifications and behavior
- Data structures and state management
- Security implementations and mechanisms
- External integrations and dependencies

1.3 Contract Address

Contract will be deployed on BNB Chain at: [To be determined after deployment]

---

2. Contract Overview

AlphaXCoinEnhanced is an ERC20-compliant token with extended functionality for implementing a comprehensive tokenomics model. The contract incorporates multiple features designed to create a secure, sustainable ecosystem.

2.1 Key Features

- **Standard Compliance**: ERC20 and ERC20Permit implementation
- **Multi-Phase Sale**: Three-tiered pricing structure with automated phase transitions
- **Reward Mechanisms**: Promotional codes, staking, referrals, and revenue sharing
- **Access Control**: Role-based permissions with KYC and blacklist functionality
- **Security Features**: Emergency mode, pausability, reentrancy protection
- **Price Oracle Integration**: Chainlink BNB/USD price feed for accurate pricing

2.2 Contract Inheritance

The contract inherits from multiple OpenZeppelin contracts:

- `ERC20`: Standard token implementation
- `ERC20Permit`: Gasless approval functionality
- `Ownable`: Ownership control
- `AccessControl`: Role-based permissions
- `Pausable`: Emergency pause capability
- `ReentrancyGuard`: Protection against reentrancy attacks

---

3. System Architecture

3.1 High-Level Architecture

The AlphaXCoin contract implements a layered architecture with distinct components:

1. **Core Token Layer**: ERC20 implementation with permit functionality
2. **Access Control Layer**: Role-based permission system
3. **Sale Management Layer**: Phase-based token distribution
4. **Reward Layer**: Staking, referrals, and promotional mechanisms
5. **Revenue Layer**: Revenue sharing and distribution
6. **Security Layer**: Emergency controls and pause functionality
7. **Integration Layer**: Chainlink oracle integration

3.2 Wallet Structure

The contract utilizes a multi-wallet architecture with immutable addresses for different aspects of the token economy:

Wallet	Address	Purpose
OWNER_WALLET	0xC20a9972a8865C697eA7E04a0c28444F778da9eb	Contract ownership
MULTISIG	0xC7Ef02A1b8A339674BDC03de7ddCFE83321399D2	Secure fund storage
SUBSCRIPTION_WALLET	0x370c6c3860B56D6bdbCa790043FbA419D4a30c36	Public sale token allocation
FEE_WALLET	0x9cEEb9430A3837281c57f3E96Ab9cfbFd7DDc547	Transaction fee collection
REWARD_WALLET	0xba1d2e78fc8CAc97B4772E5D52EBd79341C621f8	Staking and referral rewards
LIQUIDITY_WALLET	0x1ac20b0DdbB3cb1DECa51d5E9C6ce5139A423F0F	Liquidity provision
PRIVATE_WALLET	0x47aE82ADfE5a4A9ec37cE369393D53B33854AEBB	Private sale allocation
FUTURE_WALLET	0xBaC81574dDC152Ed4D77007D3cbe58e37cB9FC55	Future development funding
BURN_WALLET	0x1F2154D4BD1C91ad1F77043aF15A814f5CD84a38	Token burning

3.3 Flow Diagram

The main operational flows within the contract:

1. **Token Purchase Flow**:
2. `User (sends BNB) → Contract → Fee Collection → Phase Validation → Token Calculation → Optional Locking → Transfer → Promo/Referral Processing`
3. **Staking Flow**:

   `User (with KYC) → Stake → 180-day Lock → Redemption → Principal Return + Rewards`
4. **Revenue Sharing Flow**:

   `Admin → Distribute Revenue → Create Distribution → Users Claim → Revenue Transfer`
5. **Referral Flow**:
6. `Referrer (with KYC) → New User Purchase → Referral Reward → 180-day Vesting → Claim → Reward Transfer`

---

4. Technical Dependencies

4.1 External Libraries

The contract relies on the following external libraries:

Library	Version	Purpose
@openzeppelin/contracts/token/ERC20/ERC20.sol	4.9.0	Base ERC20 implementation
@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol	4.9.0	Permit functionality
@openzeppelin/contracts/access/Ownable.sol	4.9.0	Ownership control
@openzeppelin/contracts/access/AccessControl.sol	4.9.0	Role-based permissions
@openzeppelin/contracts/security/Pausable.sol	4.9.0	Pause functionality
@openzeppelin/contracts/security/ReentrancyGuard.sol	4.9.0	Reentrancy protection
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	4.9.0	Safe token operations
@openzeppelin/contracts/utils/math/Math.sol	4.9.0	Safe math operations

4.2 External Integrations

4.2.1 Chainlink Price Feed

The contract integrates with Chainlink's BNB/USD price feed to accurately convert between BNB and USD:

- **Feed Address**: 0x0567F2323251f0Aab15c8dFb1967E4e8A7D42aeE (BNB/USD on BNB Chain)
- **Interface**: IPriceFeed
- **Functions Used**:
  - `latestRoundData()`: Gets the latest price data
  - `decimals()`: Gets the decimal precision of the feed

The feed provides the USD price of BNB with 8 decimals of precision, which is then converted to 18 decimals for internal calculations.

4.3 Compiler Requirements

- **Solidity Version**: ^0.8.24
- **Optimizer**: Enabled (200 runs)
- **EVM Version**: London
- **License**: MIT

---

5. Access Control System

5.1 Role Definitions

The contract implements a role-based access control system with the following roles:

Role	Identifier	Purpose
KYC_ROLE	keccak256("KYC_ROLE")	Addresses that have completed KYC verification
BLACKLIST_ROLE	keccak256("BLACKLIST_ROLE")	Addresses that are prohibited from using the contract
ADMIN_ROLE	keccak256("ADMIN_ROLE")	Administrative addresses with elevated privileges
DEFAULT_ADMIN_ROLE	AccessControl default	Super-admin role for role management

5.2 Role-Based Function Access

Function Group	Required Role	Description
Staking Operations	KYC_ROLE	Functions for staking and redeeming tokens
Reward Claims	KYC_ROLE	Functions for claiming promotional and referral rewards
Revenue Claims	KYC_ROLE	Functions for claiming revenue shares
Administrative Operations	ADMIN_ROLE	Functions for contract configuration and management
Role Management	DEFAULT_ADMIN_ROLE	Functions for assigning and revoking roles

5.3 Modifiers

The contract implements the following access control modifiers:

- **onlyKYC**: Requires the caller to have KYC_ROLE and not have BLACKLIST_ROLE
- **notBlacklisted**: Requires the caller to not have BLACKLIST_ROLE
- **notInEmergency**: Requires emergency mode to be disabled
- **validPhase**: Requires the sale to be active

---

6. Data Structures

6.1 Phase Structure

The `Phase` struct represents a token sale phase:

```solidity
struct Phase {
    uint256 priceUSD;  // Price in USD with 6 decimals ($0.017 = 17000)
    uint256 cap;
    uint256 sold;
    uint256 endTime;
    uint256 leftover;
}
```

6.2 Promotional Code Structure

The `Promo` struct represents a promotional code configuration:

```solidity
struct Promo {
    uint256 min;     // Minimum purchase amount
    uint256 max;     // Maximum purchase amount
    uint256 bonus;   // Bonus token amount
    bool active;     // Whether the code is active
}
```

6.3 State Variables

6.3.1 Token Configuration

```solidity
uint256 public constant MAX_SUPPLY     = 2_500_000_000 * 1e18;   // 2.5 billion tokens
uint256 public constant MIN_PURCHASE   = 100 * 1e18;             // 100 token minimum
uint16 public burnBP                   = 10;                     // 0.1% burn rate
bool public burnEnabled;                                         // Burn toggle
uint16 public apy                      = 700;                    // 7% APY (in basis points)
```

6.3.2 Sale Configuration

```solidity
uint256 public constant TX_FEE_MIN_USD = 5 * 1e6;                // $5 minimum fee
uint16 public constant FEE_BP_PHASE1   = 75;                     // 0.75% fee
uint16 public constant FEE_BP_PHASE2   = 100;                    // 1.00% fee
uint16 public constant FEE_BP_PHASE3   = 125;                    // 1.25% fee
uint256 public constant PHASE1_END     = 1748649600;             // May 31, 2025
uint256 public constant PHASE2_END     = 1750569600;             // Jun 21, 2025
uint256 public constant PHASE3_END     = 1752950400;             // Jul 18, 2025
```

6.3.3 Time Locks

```
solidity
uint256 public constant PROMO_EXPIRY   = 40 days;              // Promo
claim window
uint256 public constant STAKING_PERIOD = 180 days;             // Staking
duration
uint256 public constant PHASE1_LOCK    = 80 days;              // Phase 1
lock period
uint256 public constant REWARD_LOCK    = 180 days;             // Reward
lock period
```

6.3.4 Cap Limitations

```
solidity
uint256 public constant PROMO_CAP      = 250_000 * 1e18;       // Promo
token cap
uint256 public constant REFERRAL_CAP   = 5_000_000 * 1e18;     //
Referral token cap
```

---

7. Token Sale Implementation

7.1 Phase Configuration

The contract implements a three-phase token sale with increasing prices:

Phase	Price (USD)	Allocation	End Date
Phase 1	$0.017	150,000,000 AXC	May 31, 2025
Phase 2	$0.027	250,000,000 AXC	June 21, 2025
Phase 3	$0.037	425,000,000 AXC	July 18, 2025

7.2 Phase Transition Logic

Phases automatically advance when either:

1. The current phase end time is reached
2. The phase allocation is fully sold

The transition process:

1. Remaining funds are sent to the MULTISIG wallet
2. Current phase is incremented
3. Unsold tokens from the previous phase are added to the next phase

7.3 Token Purchase Process

The token purchase flow in `buyWithBNB`:

1. Collect transaction fee based on current phase
2. Validate referrer (if provided)
3. Check and potentially advance phase if needed
4. Calculate token amount based on BNB amount and current USD price
5. Validate minimum purchase and phase cap
6. Update phase sold amount
7. For Phase 1, lock 50% of tokens for 80 days
8. Transfer tokens to buyer
9. Apply burn if enabled
10. Process promotional code bonus if applicable
11. Process referral reward if applicable

7.4 Fee Structure

Transaction fees increase with each phase:

Phase	Fee Percentage	Minimum Fee
Phase 1	0.75%	$5 USD
Phase 2	1.00%	$5 USD
Phase 3	1.25%	$5 USD

Fee calculation:

1. Convert transaction value from BNB to USD using Chainlink
2. Calculate percentage fee based on phase
3. Apply minimum fee if percentage fee is lower
4. Convert fee from USD to BNB
5. Send fee to FEE_WALLET

---

8. Reward Mechanisms

8.1 Promotional System

8.1.1 Promotional Codes

The contract includes five predefined promotional codes:

Promo Code	Purchase Range (AXC)	Bonus (AXC)
Mo3AXC9cl	4,500 - 12,000	500
Cl1AXC3me	15,000 - 45,000	2,000
Ah5AXC10ng	75,000 - 140,000	5,000
Kh15AXC20bo	215,000 - 285,000	15,000
Ha25AXC40bo	360,000 - 570,000	30,000

8.1.2 Promotional System Logic

1. When a user purchases tokens, they can provide a promo code
2. If the code is valid and the purchase amount is within range:
    - The bonus amount is locked for 180 days
    - The user is marked as having used that promo code
    - The total promotional allocation is updated
3. Users can claim their promotional bonuses after the expiry period

8.2 Staking System

8.2.1 Staking Configuration

- **APY**: 7% (700 basis points)
- **Staking Period**: 180 days (fixed)
- **Early Withdrawal**: Not allowed

8.2.2 Staking Process

1. User with KYC role can stake tokens using the `stake` function
2. Tokens are transferred to the contract and locked for 180 days
3. Staking time and amount are recorded
4. After the staking period, user can redeem principal and rewards using `redeemStake`
5. Rewards are calculated as: `stake_amount * apy / 10000`
6. Both principal and rewards are transferred to the user

8.3 Referral System

8.3.1 Referral Configuration

- **Reward Rate**: 5% of the referred purchase
- **Vesting Period**: 180 days
- **Total Cap**: 5,000,000 AXC

8.3.2 Referral Process

1. When a user purchases tokens, they can specify a referrer address
2. If the referrer has KYC role:

    - Referral reward (5% of purchase) is calculated
    - Reward is added to referrer's pending rewards
    - Vesting time is recorded if first referral
    - Rewards are locked for 180 days
3. After the vesting period, referrer can claim rewards using `claimReferral`

8.4 Revenue Sharing

8.4.1 Revenue Distribution Logic

1. Admin can distribute revenue using `distributeRevenue`
2. Each distribution is assigned a unique index
3. Distribution amount and total staked amount are recorded
4. Stakers can claim their share using `claimRevenue`
5. Share is calculated as: `user_stake * pool_amount / total_staked`
6. Each distribution must be claimed individually

---

9. External Integrations

9.1 Chainlink Price Feed

9.1.1 Integration Purpose

The contract uses Chainlink's BNB/USD price feed to:

1. Convert BNB to USD for accurate token pricing
2. Calculate transaction fees in USD equivalent
3. Ensure consistent token pricing regardless of BNB volatility

9.1.2 Price Feed Implementation

```
solidity
function getBNBUSDPrice() public view returns (uint256) {
    (
        uint80 roundID,
        int256 price,
        ,
        uint256 timeStamp,
        uint80 answeredInRound
    ) = IPriceFeed(PRICE_FEED).latestRoundData();

    if (price <= 0) revert InvalidPrice();
    if (timeStamp < block.timestamp - PRICE_STALE_THRESHOLD) revert
StalePrice();
```

```
        if (answeredInRound < roundID) revert PriceDataError();

        // Adjust to 18 decimals (Chainlink BNB/USD has 8 decimals)
        uint8 decimals = IPriceFeed(PRICE_FEED).decimals();
        if (decimals < 18) {
            return uint256(price) * 10**(18 - decimals);
        } else {
            return uint256(price) / 10**(decimals - 18);
        }
    }
```

9.1.3 Price Data Validation

The contract includes three validations for price data:

1. **Negative Price Check**: Ensures price is positive
2. **Staleness Check**: Ensures price is updated within the last hour
3. **Round Completeness Check**: Ensures the answer is from a complete round

---

10. Function Specifications

10.1 Constructor

```
solidity
constructor()
```

Functionality:

- Initializes the contract with ERC20 name "Alpha X Coin" and symbol "AXC"
- Sets OWNER_WALLET as the owner
- Grants necessary roles to OWNER_WALLET
- Mints initial token allocations
- Sets up sale phases and promotional codes
- Locks REWARD_WALLET tokens

10.2 Price and Conversion Functions

10.2.1 getBNBUSDPrice

```
solidity
function getBNBUSDPrice() public view returns (uint256)
```

Parameters: None
Returns: Current BNB/USD price with 18 decimals
Functionality: Gets the latest BNB/USD price from Chainlink with validation

10.2.2 convertUSDToBNB

```solidity
function convertUSDToBNB(uint256 usdAmount) public view returns (uint256)
```

Parameters:

- usdAmount: USD amount with 6 decimals

Returns: Equivalent BNB amount with 18 decimals
Functionality: Converts USD to BNB based on current price

10.2.3 calculateTokenAmount

```solidity
function calculateTokenAmount(uint256 bnbPaid, uint256 usdPrice) public
view returns (uint256)
```

Parameters:

- bnbPaid: Amount of BNB paid
- usdPrice: USD price per token with 6 decimals

Returns: Token amount with 18 decimals
Functionality: Calculates token amount based on BNB paid and USD price

10.3 Token Sale Functions

10.3.1 buyWithBNB

```solidity
function buyWithBNB(string memory code, address referrer) external payable
whenNotPaused notBlacklisted validPhase notInEmergency
```

Parameters:

- code: Promotional code
- referrer: Address of referrer

Functionality:

- Collects transaction fee
- Validates referrer
- Processes token purchase
- Applies phase locking if applicable
- Processes promotional code if valid
- Processes referral reward if applicable

10.4 Promotional Functions

10.4.1 claimPromo

```
solidity
function claimPromo(string memory code) external payable whenNotPaused
onlyKYC notInEmergency
```

Parameters:

- code: Promotional code used during purchase

Functionality:

- Validates the user has used the promo code
- Checks expiry period
- Transfers promotional bonus tokens to user

10.5 Staking Functions

10.5.1 stake

```
solidity
function stake(uint256 amount) external payable whenNotPaused onlyKYC
notInEmergency
```

Parameters:

- amount: Amount of tokens to stake

Functionality:

- Transfers tokens from user to contract
- Records staking amount and time
- Updates total staked amount

10.5.2 redeemStake

```
solidity
function redeemStake() external payable whenNotPaused onlyKYC
notInEmergency
```

Parameters: None
Functionality:

- Validates staking period completion
- Calculates reward amount
- Returns principal and rewards to user
- Updates total staked amount

10.6 Referral Functions

10.6.1 claimReferral

```
solidity
function claimReferral() external payable whenNotPaused onlyKYC
notInEmergency
```

Parameters: None
Functionality:

- Validates vesting period completion
- Transfers referral rewards to user
- Updates total referral amount

10.7 Revenue Sharing Functions

10.7.1 distributeRevenue

```
solidity
function distributeRevenue(uint256 amount) external onlyRole(ADMIN_ROLE)
notInEmergency
```

Parameters:

- amount: Amount of tokens to distribute

Functionality:

- Creates a new revenue distribution
- Records distribution amount and total staked amount
- Increments revenue index

10.7.2 claimRevenue

```
solidity
function claimRevenue(uint256 idx) external payable whenNotPaused onlyKYC
notInEmergency
```

Parameters:

- idx: Index of the revenue distribution

Functionality:

- Validates user hasn't claimed this distribution
- Calculates user's share
- Transfers share to user

10.8 Administrative Functions

10.8.1 pause and unpause

```solidity
function pause() external onlyRole(ADMIN_ROLE)
function unpause() external onlyRole(ADMIN_ROLE)
```

Parameters: None
Functionality: Pauses or unpauses contract functions

10.8.2 Emergency Mode Functions

```solidity
function activateEmergency() external onlyRole(ADMIN_ROLE)
function deactivateEmergency() external onlyRole(ADMIN_ROLE)
```

Parameters: None
Functionality: Activates or deactivates emergency mode

10.8.3 finalizePhase

```solidity
function finalizePhase() external onlyRole(ADMIN_ROLE)
```

Parameters: None
Functionality:

- Flushes current phase funds to MULTISIG
- Ends the sale immediately

10.8.4 withdrawBNB

```solidity
function withdrawBNB(address to) external onlyRole(ADMIN_ROLE)
```

Parameters:

- to: Recipient address

Functionality: Withdraws all BNB from contract to specified address

10.8.5 rescueTokens

```solidity
function rescueTokens(address token, address to, uint256 amount) external
onlyRole(ADMIN_ROLE)
```

Parameters:

- token: Address of the token contract
- to: Recipient address
- amount: Amount of tokens to rescue

Functionality: Rescues ERC20 tokens accidentally sent to contract

---

11. Events

The contract emits the following events:

11.1 Token Events

```solidity
event TokensPurchased(address indexed buyer, uint256 amount, uint8 indexed phase);
event BurnToggled(bool enabled);
```

11.2 Reward Events

```solidity
event PromoAwarded(address indexed user, string indexed code, uint256 bonus);
event StakeStarted(address indexed user, uint256 amount);
event StakeEnded(address indexed user, uint256 principal, uint256 reward);
event ReferralGranted(address indexed referrer, uint256 amount);
event ReferralClaimed(address indexed user, uint256 amount);
```

11.3 Revenue Events

```solidity
event RevenueAdded(uint256 indexed idx, uint256 amount);
event RevenueClaimed(address indexed user, uint256 indexed idx, uint256 share);
```

11.4 Administrative Events

```solidity
event PhaseFundsSent(uint8 indexed phase, uint256 amount);
event PhaseAdvanced(uint8 indexed oldPhase, uint8 indexed newPhase);
event SaleEnded();
event APYUpdated(uint16 oldAPY, uint16 newAPY);
event TokensLocked(address indexed user, uint256 amount, uint256 unlockTime);
```

11.5 Access Control Events

```solidity
event KYCGranted(address indexed user);
event KYCrevoked(address indexed user);
event BlacklistAdded(address indexed user);
event BlacklistRemoved(address indexed user);
event EmergencyModeActivated(address indexed by);
event EmergencyModeDeactivated(address indexed by);
```

---

12. Security Implementation

12.1 Access Control Implementation

The contract uses OpenZeppelin's AccessControl for role-based permissions:

- `DEFAULT_ADMIN_ROLE`: Super-admin role for role management
- `KYC_ROLE`: Required for staking and claiming rewards
- `BLACKLIST_ROLE`: Prohibits participation in the ecosystem
- `ADMIN_ROLE`: Allows administrative operations

12.2 Reentrancy Protection

The contract uses OpenZeppelin's ReentrancyGuard to protect against reentrancy attacks. All functions that transfer tokens or ether implement the `nonReentrant` modifier.

12.3 Emergency Circuit Breakers

The contract implements two emergency mechanisms:

1. **Pausable**: Pauses most contract functions
2. **Emergency Mode**: More restrictive than pause, blocks most operations

12.4 Input Validation

The contract implements comprehensive input validation:

- Custom error types for specific validation failures
- Early validation of inputs and state preconditions
- Oracle data validation for price feed

12.5 Fund Security

The contract implements several fund security measures:

- Immutable wallet addresses
- Multi-signature wallet for main funds
- Token locking mechanisms
- Owner-restricted fund movement

12.6 Oracle Security

Price feed security measures:

- Staleness check (1-hour threshold)
- Negative price check
- Round completeness check

---

13. Gas Optimization

The contract implements several gas optimization techniques:

13.1 Custom Errors

Uses custom error types instead of revert strings for gas efficiency:

```
solidity

error Unauthorized(); error InsufficientFee(uint256,uint256); error SaleInactive();

error BelowMin(uint256); error PhaseCap(); error LockedTokens(); error InvalidReferrer();

error NoPromo(); error NothingToClaim(); error ZeroAddress(); error EmergencyMode();

error PriceDataError(); error StalePrice(); error InvalidPrice();
```

13.2 Storage Optimization

Efficient use of storage slots:

- Appropriate uint sizes (uint16 for percentages)
- Constant variables for fixed values

13.3 Function Visibility

Appropriate function visibility modifiers:

- Internal functions for internal logic
- Public only when external access is required

13.4 Logic Optimization

- Early returns and short-circuiting
- Efficient conditional evaluations
- Minimal redundant calculations

---

14. Deployment Requirements

14.1 Deployment Environment

- **Network**: BNB Chain Mainnet
- **Compiler Version**: Solidity 0.8.24
- **Optimization**: Enabled (200 runs)

14.2 Deployment Wallet Requirements

- High security wallet (hardware wallet recommended)
- Sufficient BNB for deployment gas costs
- Access to all wallet addresses specified in the contract

14.3 Pre-Deployment Checklist

1. Verify all immutable wallet addresses
2. Confirm phase timing and pricing
3. Validate promotional code setup
4. Ensure Chainlink price feed address is correct
5. Verify token allocation percentages

14.4 Post-Deployment Configuration

1. Verify contract is initialized correctly
2. Grant necessary roles to operating wallets
3. Verify token balances in designated wallets
4. Test basic functionality (purchase, stake, etc.)
5. Verify contract on BscScan

---

15. Testing Guidelines

15.1 Unit Tests

Recommended unit tests for core functionality:

- Token transfer and approval
- Role-based access control

- Phase transition logic
- Price conversion accuracy
- Fee calculation correctness
- Staking and reward calculation
- Revenue sharing distribution
- Promotional code validation

15.2 Integration Tests

Recommended integration tests:

- Full token purchase flow
- Staking and redemption cycle
- Referral reward flow
- Revenue distribution and claiming
- Chainlink price feed integration

15.3 Security Tests

Recommended security tests:

- Access control boundary testing
- Reentrancy attack simulation
- Price manipulation attack simulation
- Emergency mode activation and deactivation
- Pause functionality verification
- Locked token transfer attempts

---

16. Appendices

16.1 Error Codes

Error Code	Description
Unauthorized	Caller doesn't have required permission
InsufficientFee	Insufficient BNB sent to cover fee
SaleInactive	Token sale is not active
BelowMin	Purchase below minimum amount
PhaseCap	Phase allocation cap reached
LockedTokens	Attempted to transfer locked tokens
InvalidReferrer	Invalid referrer address
NoPromo	Promotional code not valid or used
NothingToClaim	No rewards to claim
ZeroAddress	Zero address provided

Error Code	Description
EmergencyMode	Function disabled in emergency mode
PriceDataError	Error in price feed data
StalePrice	Price data is outdated
InvalidPrice	Price is invalid (zero or negative)

16.2 Glossary

- **APY**: Annual Percentage Yield
- **KYC**: Know Your Customer
- **AML**: Anti-Money Laundering
- **Basis Points (BP)**: 1/100th of a percent (10000 BP = 100%)
- **Oracle**: External data provider (Chainlink)
- **Reentrancy**: Attack vector where a function is called recursively

---