



Alireza Yousefian
Github: [itsnyx](#)



Mohammad Hossein Pour
Github: [M-hoseinpour](#)



Hossein Hamzehzadeh
Github: [Hamzeh01](#)

SMART CONTRACTS

REFERENCES & PREREQUERES

- [A Comprehensive Introduction to Blockchain](#)[Github]
- [WEB3 Swap](#)[Github]

INTRODUCTION TO SMART CONTRACTS

- Brief explanation of smart contracts 📱
- Importance in blockchain technology 🔗



ACCESS CONTROL IN SMART CONTRACT

Access control with ownable contract

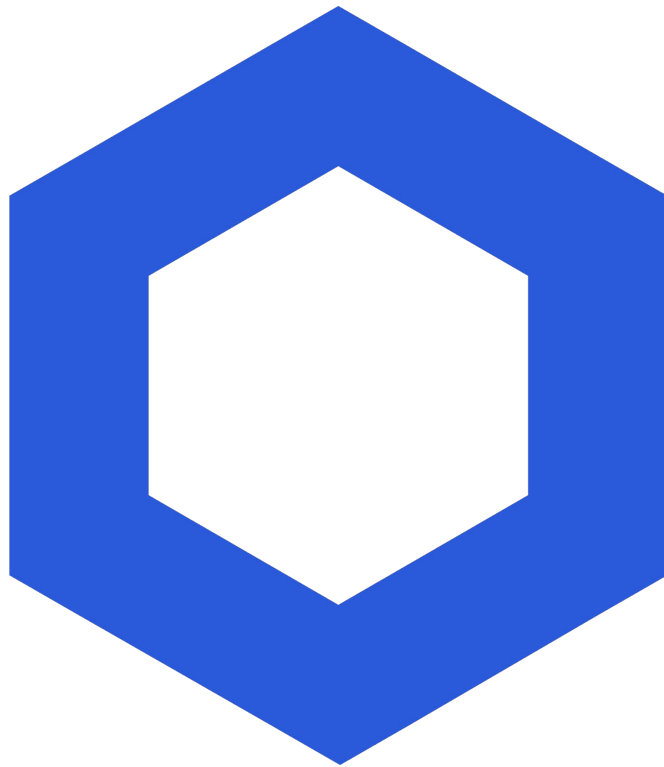
- Overview of the **Ownable** contract.🔨
- Explanation of the **_owner** variable and **onlyOwner** modifier.
🕒
- Demonstration of how **ownership** can be transferred.🚚

OWNABLE

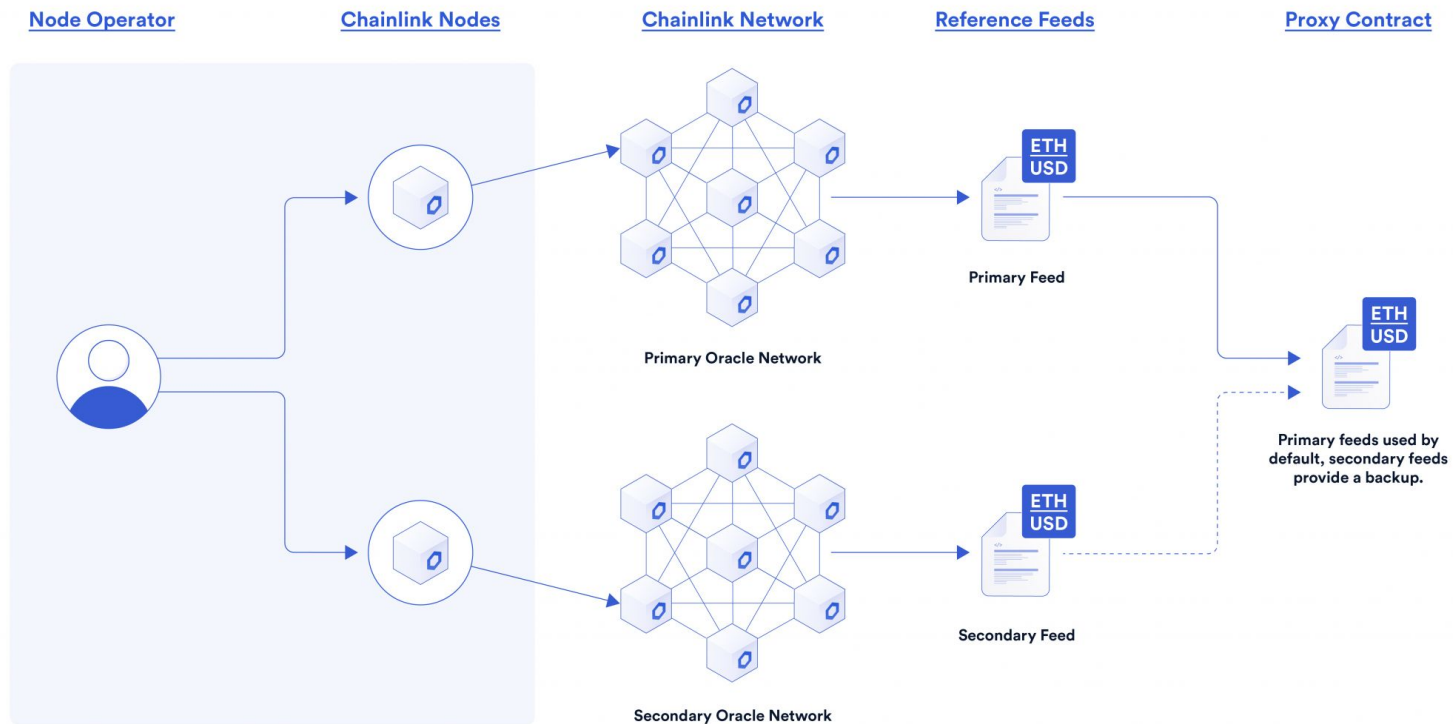
CONTRACTS

INTEGRATING EXTERNAL DATA WITH CHAINLINK




- Introduction to **Chainlink** and its role in providing decentralized oracle services.
- Explanation of the **DataFeeds** contract.
- Mention of the **AggregatorV3Interface** for price feed interaction.



PRICE FEED IMPLEMENTATION

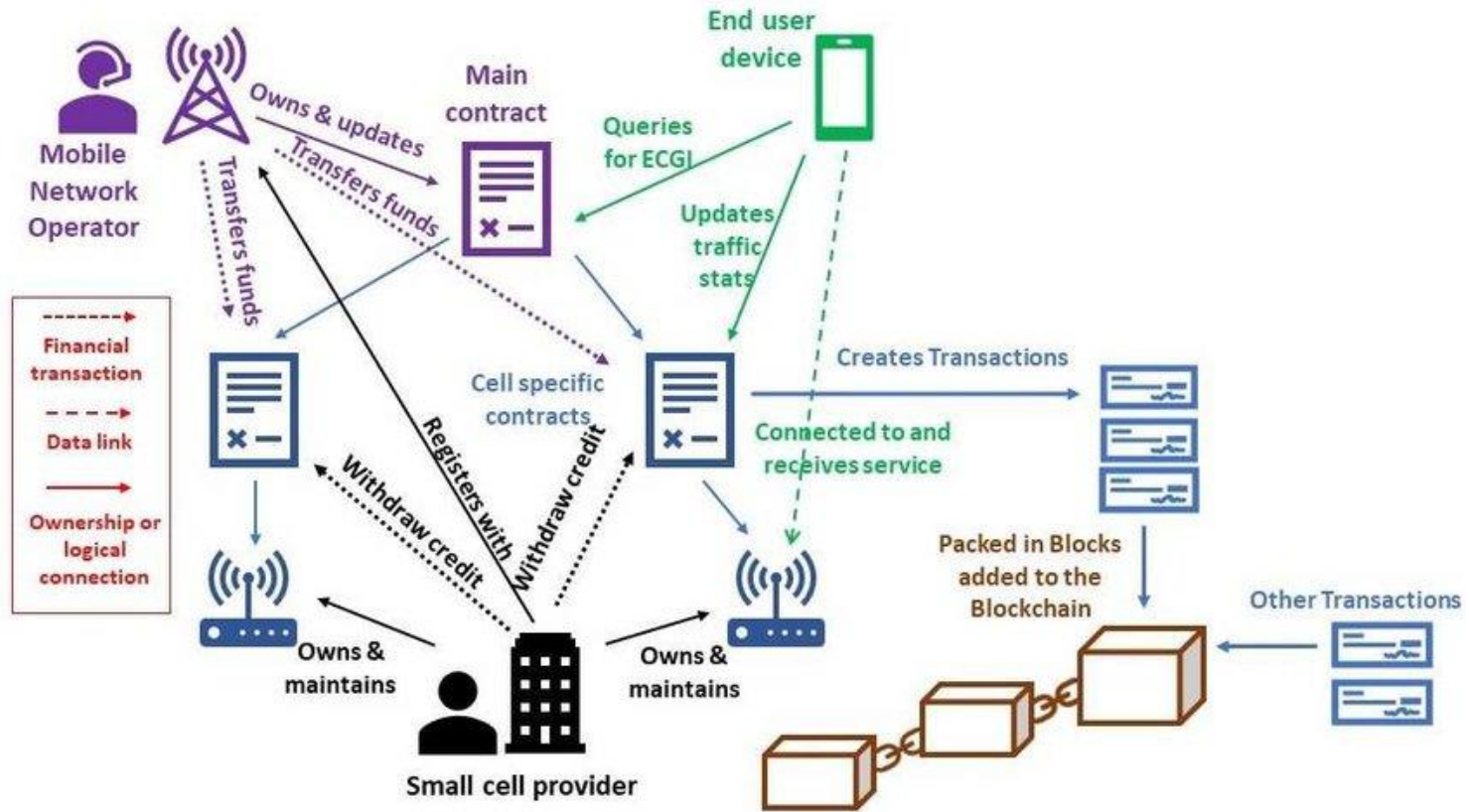


UTILIZING PRICE FEED IN TOKEN PRESALE

- How the price feed interacts with the Token Presale contract. 
- Importance of accurate pricing in token-related operations. 
- The role of the **getBNBLatestPrice** function. 

SYSTEM ARCHITECTURE OVERVIEW

Overall system architecture






OVERALL SYSTEM ARCHITECTURE

SECURITY CONSIDERATIONS

Ensuring Security in Smart Contracts

SMART CONTRACT SECURITY FEATURES

TokenPresale Contract

- Access Control 
- Admin Functionality 
- Activation/Deactivation Mechanism 
- Modifiers for Admin Functions








Ownable Contract

- Ownership Transfer
- OnlyOwner Modifier

DataFeeds Contract

- Immutable Price Feed Address
- Internal Function for Price Retrieval 

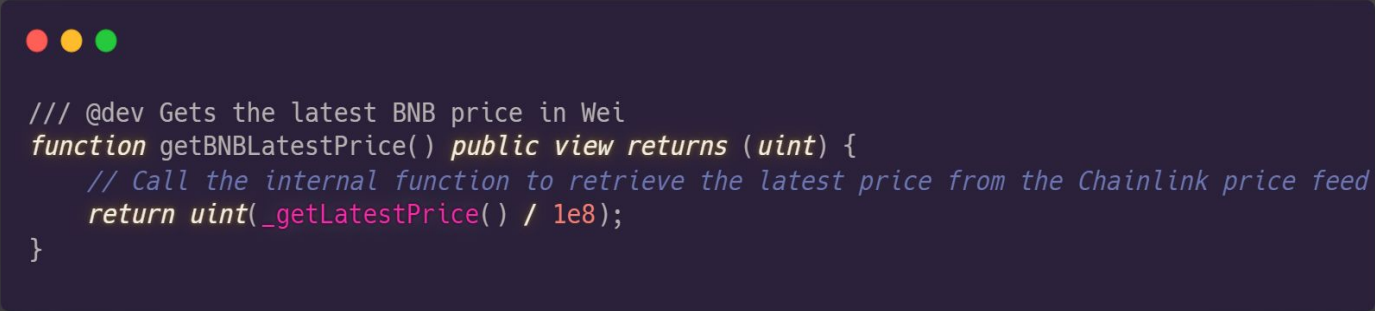
BEST PRACTICES FOR SECURE SMART CONTRACT DEVELOPMENT

- Immutable Contracts 
- Access Control 
- Use of Modifiers 
- External Dependency Considerations 
- Testing and Auditing 
- Gas Limit Considerations 
- Upgradeability Considerations 

WEB3 PRESALE

RETRIEVING THE LATEST BNB PRICE

- The function calls the internal `_getLatestPrice` function to fetch the most recent price data.
- The result is then converted to Wei by dividing it by `1e8`, ensuring compatibility with Ethereum's base unit.



```
/// @dev Gets the latest BNB price in Wei
function getBNBLatestPrice() public view returns (uint) {
    // Call the internal function to retrieve the latest price from the Chainlink price feed
    return uint(_getLatestPrice() / 1e8);
}
```


PARTICIPATING IN THE TOKEN PRESALE

- Checks if the presale is currently active and if the participant's wallet is allowed. 🚫
- If it's the participant's first entry, calculates token amount from provided BNB and updates allocations. 📊
- If the participant has already entered, calculates additional token amount, checks if the amount is acceptable, and updates allocations. ✓
- **Purpose:** This function allows participants to enter the token presale, checking their previous participation and updating token allocations accordingly.

PARTICIPATING IN THE TOKEN PRESALE

```

function enterPresale() public payable {
    // Ensure the presale is currently active
    require(isActive, "TokenPresale: Presale is currently not active.");

    // Ensure the participant's wallet is in the allowed list
    require(isAllowed(msg.sender), "TokenPresale: Access Denied, Your Wallet Should be in Allowed List");

    if (getWalletTokenParticipation(msg.sender) == 0) {
        // Participant is entering for the first time
        require(checkBNBAmount(msg.value), "TokenPresale: Amount is not valid");

        // Calculate token amount from BNB
        uint amount = getTokenAmountFromBNB(msg.value);

        // Update participant's token and BNB allocations
        walletTokenAllocation[msg.sender] = amount;
        walletBNBAllocation[msg.sender] = msg.value;

        // Update totalTokenSale and totalBNBValue
        totalTokenSale += amount;
        totalBNBValue += msg.value;

        // Send tokens to participant's wallet
        sendTokenToWallet(amount, msg.sender);
    }
}
```

PARTICIPATING IN THE TOKEN PRESALE

```
else {  
    // Participant has already entered, update the amount  
  
    // Calculate token amount from additional BNB  
    uint amount = getTokenAmountFromBNB(msg.value);  
  
    // Check if the provided amount is acceptable  
    require(checkParticipationUpdate(amount), "TokenPresale: Amount Provided is not acceptable.");  
  
    // Update participant's token and BNB allocations  
    walletTokenAllocation[msg.sender] += amount;  
    walletBNBAllocation[msg.sender] += msg.value;  
  
    // Update totalTokenSale and totalBNBValue  
    totalTokenSale += amount;  
    totalBNBValue += msg.value;  
  
    // Send additional tokens to participant's wallet  
    sendTokenToWallet(amount, msg.sender);  
}
```

ADMIN ACCESS CONTROL

- The **modifier** checks if the caller is either the admin or the owner of the smart contract.
- If the condition is met, the function continues **execution**; otherwise, an **error** is thrown.

```
/// @dev Modifier to restrict function access to the admin or owner.  
modifier onlyAdmin() {  
    // Ensure the caller is either the admin or the owner  
    require(msg.sender == admin || msg.sender == owner(), "Caller is neither admin nor  
owner");  
    // Continue with the function if the condition is met  
}
```

TOKEN WITHDRAWAL FUNCTIONALITY

- The function takes two parameters: **amount** (the quantity of tokens to withdraw) and **beneficiary_** (the address to receive the withdrawn tokens).
- The withdrawal is contingent on the successful transfer of tokens to the specified beneficiary.

```
/// @dev Withdraws tokens from the contract and transfers them to the specified beneficiary.
/// @param amount Amount of tokens to withdraw.
/// @param beneficiary_ Destination address for the withdrawn tokens.
function withdrawToken(uint256 amount, address beneficiary_) public onlyOwner {
    // Ensure the transfer of tokens to the beneficiary is successful
    require(token().transfer(beneficiary_, amount));
}
```

BNB WITHDRAWAL FUNCTIONALITY

- The function takes a parameter **to** (the address to receive the withdrawn BNB).
- It ensures that the destination address is valid and not zero.
- It checks that there is a positive BNB balance in the contract before initiating the transfer.

```
/// @dev Withdraws BNB from the contract and transfers it to the specified destination address.
/// @param to Destination address to receive the withdrawn BNB.
function withdrawBNB(address payable to) public payable onlyOwner {
    // Ensure the destination address is not zero
    require(to != address(0), "Destination address is zero");

    // Get the current balance of BNB in the contract
    uint Balance = address(this).balance;

    // Ensure there is a positive balance to withdraw
    require(Balance > 0 wei, "Error! No balance to withdraw");

    // Transfer the entire balance to the specified destination address
    to.transfer(Balance);
}
```

CONCLUSION AND Q&A

Wrapping Up and Questions

THANK YOU!