Description:

The contract implements Presale functionality for a specific token (myToken is our testcase). The presale sets a hard cap on the total amount of sale tokens to be sold. The contract gives the flexibility to buy the Token with either the whitelisted tokens (tokenA, tokenB in our testcase) or the protocol's native currency (e.g., ether or BNB).

Functions:

1.

```
function setSaleTokenParams(
        address _saleToken, uint256 _totalTokensforSale, uint256 _rate
    )external onlyOwner saleStarted{};
```

This function sets the parameters for the token to be sold in the Presale. Only the owner of the contract has the authority to call this function.

Parameters:

address _saleToken: address of the token to be sold in the Presale

uint256 _totalTokensforSale: number of tokens to be sold in the Presale (with decimals)

uint256 _rate: rate of sale token with respect to the protocol's native currency (with decimals)

2.

```
function setSalePeriodParams(
        uint256 _preSaleStartTime,
        uint256 _preSaleEndTime,
        uint256 _lockingPeriod1,
        uint256 _lockingPeriod2,
        uint256 _percentTokens1
    )external onlyOwner saleStarted saleValid(_preSaleStartTime,
_preSaleEndTime, _lockingPeriod1, _lockingPeriod2){};
```

This functions sets the Time periods for the Presale. This function can be called only by the owner of the contract and cannot be called during the Presale.

Parameters:

uint256 _preSaleStartTime: Start time for the Presale

uint256 _preSaleEndTime: End time for the presale

uint256 _lockingPeriod1: First locking period after the presale

uint256 _lockingPeriod2: Second locking period after the presale

uint256 _percentTokens1: Percentage of sale tokens available for withdrawal after 1st locking period ends

3.

```
function addWhiteListedToken(
        address[] memory _tokens,
        uint256[] memory _prices
    ) external onlyOwner saleStarted{};
```

This function allows the owner to add the tokens whitelisted with which the users can buy the sale token.

Parameters:

address[] memory _tokens: array of the addresses of the tokens to be whitelisted

uint256[] memory _prices: rates of the sale token with respect to the tokens passed in the array above (conversion done with decimals)

4.

```
function updateTokenRate(
        address[] memory _tokens,
        uint256[] memory _prices,
        uint256 _rate
    )external onlyOwner{};
```

This function allows the owner to update the rates of the sale token with respect to whitelisted tokens and protocol's native currency. This function can be anytime by the owner.

Parameters:

address[] memory _tokens: array of the addresses of the tokens to be whitelisted

uint256[] memory _prices: rates of the sale token with respect to the tokens passed in the array above

uint256 _rate: rate of sale token with respect to the protocol's native currency.

5.

```
function getTokenAmount(address token, uint256 amount)
        public view returns (uint256){};
```

The function returns the amount of sale tokens to be given to a user in exchange of "token" and its "amount".

Parameters:

address token: address of the whitelisted token

uint256 amount: amount of the token passed (with decimals)

Function description:

```
uint256 amtOut;
      if(token != address(0)){
           require(tokenWL[token] == true, "Presale: Token not whitelisted");
           // uint tokenDec = IERC20(token).decimals();
           uint256 price = tokenPrices[token];
           amtOut = amount.mul(10**saleTokenDec).div(price);
        }
        else{
           amtOut = amount.mul(10**saleTokenDec).div(rate);
        }
        return amtOut;
```

The buyer has 2 options for buying the sale token. Either through an ERC20 token, or by using the protocol's native currency (e.g. Ether)

The function receives the address of the whitelisted token and its amount. If the buyer is using native currency then address(0) is passed as the first arg.

The amount of sale token is calculated by using the conversion rate of the particular token passed.

6.

```
function buyToken(address _token, uint256 _amount) external payable
saleDuration{};
```

External function called by user to buy sale Tokens. The user passes the address and amount of the token with which he/she wants to buy the sale token. The process proceeds only when the token is whitelisted. In case it is using native currency, address(0) is passed in the _token param and no amount is passed. The amount of currency is sent in "msg.value".

Parameters:

address _token: address of the token passed

uint256 _amount: amount of token passed (with decimals)

7.

```
function withdrawToken()external {}
```

External function called by user to withdraw the sale tokens from the contract.

Function description:

The user is able to withdraw a certain percentage of tokens after the first locking period ends, and the rest of the tokens after the end of second locking period.

Consider the following example:

The 1st locking period ends on 5th march and 2nd locking period ends on 15th march, and the percentage tokens available after 1st locking period is 30%. The user holds 100 * 10^18 sale tokens.

1. The user will not be able to withdraw any amount of the sale token before 5th march
2. Between 5th and 15th march, user will be able to withdraw maximum of 0.3 * 100 * 10^18 sale tokens, i.e., 30 * 10^18
3. After 15th march, the user will be able to withdraw the rest of the 70 * 10^18 sale tokens.