SMART CONTRACT AUDIT - FRONTLINE HEROES TOKEN

Eng Carlos Adolfo Alejandro

ATEITIS INFORMATION TECHNOLOGY LLC

DECEMBER 31, 2021

Author: ATEITIS

ATEITIS INFORMATION TECHNOLOGY

Table of Contents

Confidentiality Clause

The document is property of Frontline Heroes Token and includes information that is privileged, confidential and/or cannot be disclosed. If you are not an authorized recipient, please return this document to the owner mentioned above.

This is a security audit report document and may contain information that is confidential. It includes any potential vulnerabilities and malicious codes that can be used to exploit the software. This must be referred internally and should only be available to the public once the issues have been resolved.

The disclosure, distribution, copying or use of all or part of this document to any person other than the corresponding recipient is forbidden without prior written permission of Frontline Heroes Token.

Limitations and Disclaimer

Frontline Heroes Token is a standard BEP20 token smart contract. This audit only considers Frontline Heroes Token smart contracts and does not include any other smart contracts in the platform.

The observations or conclusions expressed in this document are the result of the analysis of the information obtained from the execution of technical tests performed on the set of devices identified as part of the selected sample, therefore:

- They are only applicable to the reviewed environment under the conditions discussed above, i.e. they are not applicable to other environments;
- There may be other vulnerabilities in addition to those currently detected related to the modification of the environment under analysis.

Throughout the evaluation process described in this document, the tests have been carried out only on the addressing defined in the aforementioned SCOPE, excluding intermediate elements not included in it.

This report presents all the findings regarding the audit performed on December 31, 2021.

The purpose of this audit was to cover the following:

- Ensure that all claimed functions exist and function correctly.

- Identify any security vulnerabilities that may be present in the smart contract.

ATEITIS INFORMATION TECHNOLOGY LLC team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. It is important to consider that this report should not be relied upon alone. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Therefore, the audit cannot guarantee explicit security of the audited smart contracts.

Investors Advice: the technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Scope and Methodology

Name	**Frontline Heroes Token**
Platform	**BSC / Solidity**
File	FHT.sol
File MD5 Hash	7CA2328D7FFD371D393AF3945458DED1
Audit Date	December 31, 2021

The goals of our security audits are to improve the quality of systems we review We use the following methodology in our security audit process:

Manual Code Review

We look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors and random number generators. We also look for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

We look at the project's website to get a high level understanding of what functionality the software under review provides. After that, we meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies and generally investigate details other than the implementation.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful solution. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not verified the feasibility and impact of the issue yet. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take and finally we suggest the requirements for solution engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

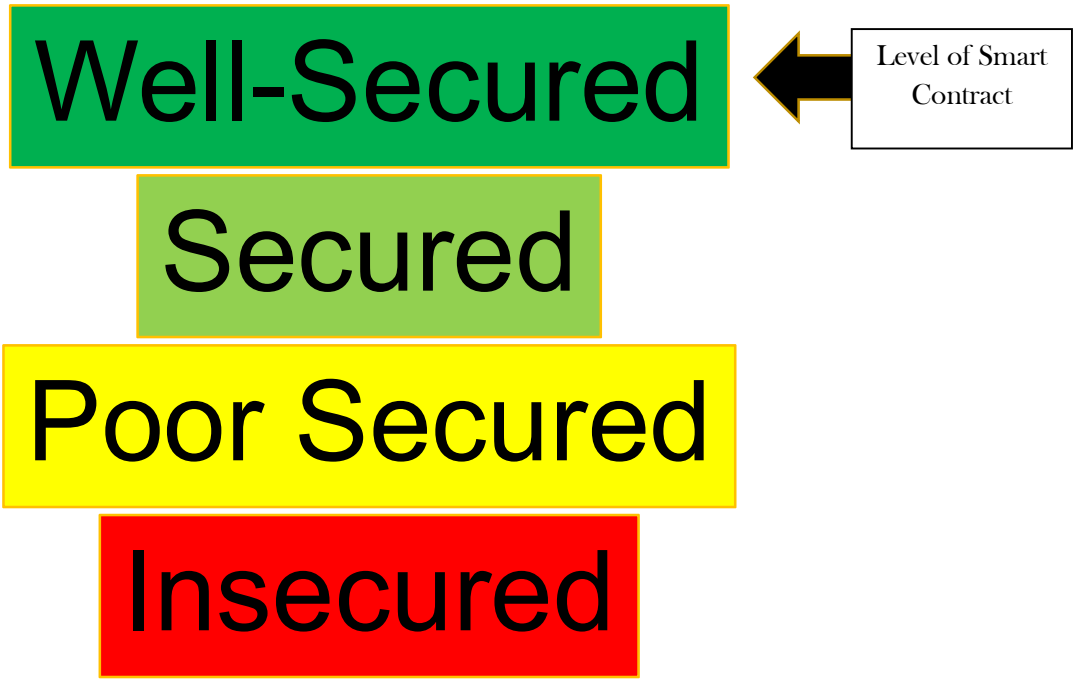
Executive Summary

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Frontline Heroes Token • Symbol: FHT • Decimals: 18 • Total Supply: 1 Quadrillion	YES, This is valid.
• Minimum Tokens Before Swap Amount: 10 Million FHT • Maximum Transaction Amount: 10 Trillion FHT • Charity Fee: 5% • Liquidity Fee: 3% • Tax Fee (Reflections to holders): 2%	YES, This is valid. Owner authorized wallet can set some percentage value and we suggest handling the private key of that wallet securely.

Overall Result

According to the standard audit assessment, FHT TOKEN (smart contracts details in scope) are "Well-Secured".



All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 2 low.

ATEITIS INFORMATION TECHNOLOGY

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Insecure Compiler Version	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Failed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Function Default Visibility	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Passed
	High consumption 'for/while' loop	Failed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Code Quality

The audit scope has 1 (one) smart contract file. Smart contract contains libraries, smart contracts, inherits and interfaces. This is a compact and well written smart contract.

The libraries in Frontline Heroes Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the Frontline Heroes Token.

The Frontline Heroes Token team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on smart contracts.

CONCLUSION: PASSED

Documentation

Frontline Heroes Token has given us a smart contract code in the form of code. The hash of that code is mentioned above in the table.

As code parts are not well commented, it is not easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

CONCLUSION: PASSED

Use of Dependencies

According to our observation, this smart contract infrastructure uses libraries that are based on well-known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

CONCLUSION: PASSED

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	lockTheSwap	modifier	Passed	No Issue
3	name	read	Passed	No Issue
4	symbol	read	Passed	No Issue
5	decimals	read	Passed	No Issue
6	totalSupply	read	Passed	No Issue
7	balanceOf	read	Passed	No Issue

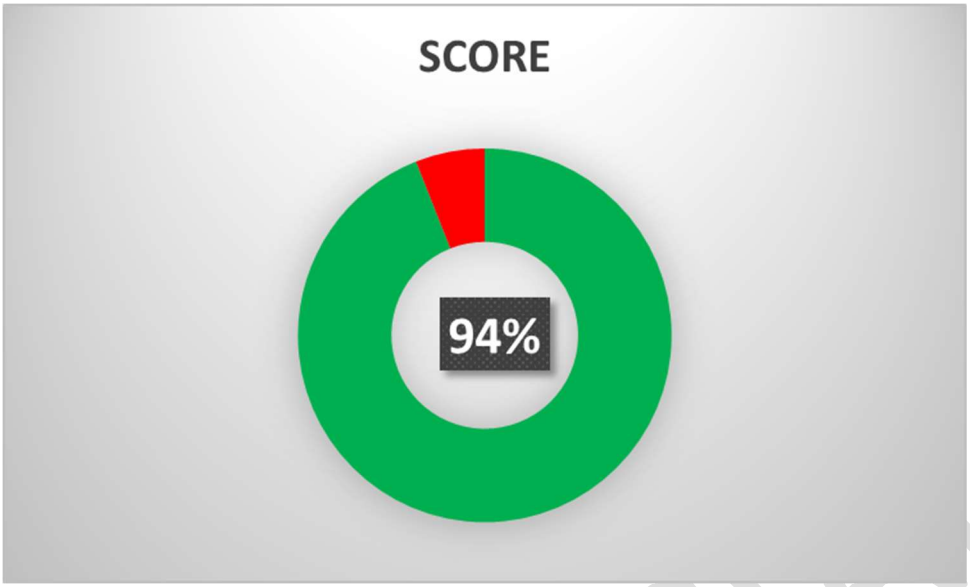
8	transfer	write	Passed	No Issue
9	allowance	read	Passed	No Issue
10	approve	write	Passed	No Issue
11	transferFrom	write	Passed	No Issue
12	increaseAllowance	write	Passed	No Issue
13	decreaseAllowance	write	Passed	No Issue
14	isExcludedFromReward	read	Passed	No Issue
15	totalFees	read	Passed	No Issue
16	minimumTokensBeforeSwapAmount	read	Passed	No Issue
17	deliver	write	Passed	No Issue
18	reflectionFromToken	read	Passed	No Issue
19	tokenFromReflection	read	Passed	No Issue
20	excludeFromReward	write	Passed	No Issue
21	includeInReward	external	Low Risk	Refer Audit Findings
22	_approve	write	Passed	No Issue
23	_transfer	write	Passed	No Issue
24	swapAndLiquify	write	Passed	No Issue
25	swapTokensForEth	write	Passed	No Issue
26	addLiquidity	write	Passed	No Issue
27	_tokenTransfer	write	Passed	No Issue
28	_transferStandard	write	Passed	No Issue
29	_transferToExcluded	write	Passed	No Issue
30	_transferFromExcluded	write	Passed	No Issue
31	_transferBothExcluded	write	Passed	No Issue
32	_reflectFee	write	Passed	No Issue
33	_getValues	write	Passed	No Issue
34	_getTValues	read	Passed	No Issue
35	_getRValues	write	Passed	No Issue
36	_getRate	read	Passed	No Issue
37	_getCurrentSupply	read	Low Risk	Refer Audit Findings
38	_takeLiquidity	write	Passed	No Issue
39	calculateTaxFee	read	Passed	No Issue
40	calculateLiquidityFee	read	Passed	No Issue
41	removeAllFee	write	Passed	No Issue
42	prepareForPreSale	external	Access only Owner	No Issue
43	afterPreSale	external	Access only Owner	No Issue
44	isExcludedFromFee	read	Passed	No Issue
45	excludeFromFee	write	Low Risk	Refer AuditFindings

46	includeInFee	write	Low Risk	Refer AuditFindings
47	setTaxFeePercent	external	Low Risk	Refer Audit Findings
48	setLiquidityFeePercent	external	Low Risk	Refer Audit Findings
49	setCharityFeePercent	external	Low Risk	Refer Audit Findings
50	setMaxTxAmount	external	Low Risk	Refer Audit Findings
51	setNumTokensSellToAddToLiquidity	external	Low Risk	Refer Audit Findings
52	setcharityWalletAddress	external	Low Risk	Refer Audit Findings
53	setSwapAndLiquifyEnabled	write	Access only Owner	No Issue
54	transferToAddressETH	write	Passed	No Issue
55	receive	external	Passed	No Issue
56	owner	read	Passed	No Issue
57	onlyOwner	modifier	Passed	No Issue
58	renounceOwnership	write	Access only Owner	No Issue
59	transferOwnership	write	Access only Owner	No Issue
60	getUnlockTime	read	Passed	No Issue
61	getTime	read	Passed	No Issue
62	lock	write	Access only Owner	No Issue
63	unlock	write	Passed	No Issue

CONCLUSION: LOW RISK

Conclusion

Taking into account the information provided by the client including the investigations and each of the analyses reported in this report, ATEITIS INFORMATION TECHNOLOGY LLC is of the opinion that the overall security status of the smart contract is "Well-Secured". It should be noted that this opinion is based on each of the aspects specified in this report. ATEITIS INFORMATION TECHNOLOGY LLC recommends mitigating the low criticality issues specified in the report, in order to increase the performance and efficiency of the report, as well as to increase the level of security in the audited environment.



Based on the result obtained from the Audit, ATEITIS INFORMATION TECHNOLOGY LLC is of the opinion that the present token complies with good security practices in general and can be deployed in different CEX.

OVERALL RESULT : "WELL SECURED".

Report of Vulnerabilities

Severity definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually easy to exploit and can lead to loss of tokens, among others.
High	High-level vulnerabilities are difficult to exploit. However, they also have significant impact on smart contract execution, e.g. public access to crucial.
Medium	Medium-level vulnerabilities are important to fix. However, they can't lead to the loss of tokens.
Low	Low-level vulnerabilities are mostly related to outdated,unused code snippets, among others,that can't have significant impact on execution.

Audit Findings

CRITICAL SEVERITY

No Critical severity vulnerabilities were found.

HIGH SEVERITY

No High severity vulnerabilities were found.

MEDIUM SEVERITY

No Medium severity vulnerabilities were found.

LOW SEVERITY

Function input parameters lack of check

Vulnerability detail

There are many functions and there are no valid input values.

Functions are:

- excludeFromFee
- includeInFee
- setTaxFeePercent
- setLiquidityFeePercent
- setCharityFeePercent
- setMaxTxAmount

- setNumTokensSellToAddToLiquidity
- setcharityWalletAddress

Solution

We suggest adding proper validation before execution.

- Address variable should not be address(0);
- Integer value must be greater than 0;
- Fee set as percentage should have some range validation.

Infinite loops possibility

Vulnerability detail

As array elements will increase, then it will cost more and more gas. And eventually, it will stop all the functionality. After several hundreds of transactions, all those functions depending on it will stop. We suggest avoiding loops. For example, use mapping to store the array index and query that data directly, instead of looping through all the elements to find an element.

Solution

Adjust logic to replace loops with mapping or other code structure:

Write Function:

- includeInReward() - _excluded.length.

Read Function:

- _getCurrentSupply() - _excluded.length.

Appendix

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble.

Following are Admin functions:

- **excludeFromReward: The Owner can check if the account is already excluded or**
- **not then exclude from reward.**
- **includeInReward: The Owner can include wallet address in reward.**
- **prepareForPreSale: The Owner can prepare an account for pre-sale.**
- **afterPreSale: The Owner can set an after sale account.**
- **excludeFromFee: The Owner can exclude a fee from the account.**
- **includeInFee: The Owner can include a fee in the account.**
- **setTaxFeePercent: The Owner can set the tax, fee percentage.**
- **setLiquidityFeePercent: The Owner can set liquidity fee percentage.**
- **setCharityFeePercent: The Owner can set charity fee percentage.**
- **setMaxTxAmount: The Owner can set the maximum tax amount.**
- **setNumTokensSellToAddToLiquidity: The Owner can set the number of tokens sold**
- **to add to the liquidity address.**
- **setcharityWalletAddress: The Owner can set a charity wallet address.**
- **setSwapAndLiquifyEnabled: The Owner can set swap and liquify enabled status**

Code

```
// SPDX-License-Identifier: Unlicensed

pragma solidity ^0.8.4;


abstract contract Context {

    function _msgSender() internal view virtual returns (address payable) {

        return payable(msg.sender);

    }


    function _msgData() internal view virtual returns (bytes memory) {

        this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691

        return msg.data;

    }

}
```

```
interface IBEP20
{
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeMath {

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;
        return c;
    }
```

```
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) { return 0; }
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }


    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        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;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}
```

```
library Address {

    function isContract(address account) internal view returns (bool) {
        // 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 != accountHash && codehash != 0x0);
    }

    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

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

    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    function functionCall(address target, bytes memory data, string memory errorMessage) internal
returns (bytes memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
    }
```

```
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns
(bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory
errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string
memory errorMessage) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
        if (success) {
            return returndata;
        } else {

            if (returndata.length > 0) {
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}
```

```
contract Ownable is Context {
    address private _owner;
    address private _previousOwner;
    uint256 private _lockTime;

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

    constructor () {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    function owner() public view returns (address) {
        return _owner;
    }

    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
```

```
    }

    function getUnlockTime() public view returns (uint256) {
        return _lockTime;
    }

    function getTime() public view returns (uint256) {
        return block.timestamp;
    }

    function lock(uint256 time) public virtual onlyOwner {
        _previousOwner = _owner;
        _owner = address(0);
        _lockTime = block.timestamp + time;
        emit OwnershipTransferred(_owner, address(0));
    }

    function unlock() public virtual {
        require(_previousOwner == msg.sender, "You don't have permission to unlock");
        require(block.timestamp > _lockTime , "Contract is locked until 7 days");
        emit OwnershipTransferred(_owner, _previousOwner);
        _owner = _previousOwner;
    }
}

// pragma solidity >=0.5.0;

interface IUniswapV2Factory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);
```

```
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
}


// pragma solidity >=0.5.0;

interface IUniswapV2Pair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
```

```
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32
s) external;

    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32
blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;
```

```
    function initialize(address, address) external;
}

// pragma solidity >=0.6.2;

interface IUniswapV2Router01 {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
```

```
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB);
function removeLiquidityETH(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external returns (uint amountToken, uint amountETH);
function removeLiquidityWithPermit(
    address tokenA,
    address tokenB,
    uint liquidity,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
) external returns (uint amountA, uint amountB);
function removeLiquidityETHWithPermit(
    address token,
    uint liquidity,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline,
    bool approveMax, uint8 v, bytes32 r, bytes32 s
```

```
    ) external returns (uint amountToken, uint amountETH);

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);

    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);

    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint
deadline)
        external
        payable
        returns (uint[] memory amounts);

    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path,
address to, uint deadline)
        external
        returns (uint[] memory amounts);

    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path,
address to, uint deadline)
        external
        returns (uint[] memory amounts);

    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint
deadline)
        external
        payable
```

```
        returns (uint[] memory amounts);


    function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);

    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint
amountOut);

    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint
amountIn);

    function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory
amounts);

    function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory
amounts);

}




// pragma solidity >=0.6.2;


interface IUniswapV2Router02 is IUniswapV2Router01 {
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountETH);
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
```

```
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
}

contract FHT is Context, IBEP20, Ownable {
    using SafeMath for uint256;
    using Address for address;
```

```
    address payable public charityWalletAddress =
payable(0x8f020DB49E4bd2C97C7C95eA9775bf84b95C576A); // charity Address

    address public immutable deadAddress = 0x000000000000000000000000000000000000dEaD;

    mapping (address => uint256) private _rOwned;

    mapping (address => uint256) private _tOwned;

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


    mapping (address => bool) private _isExcludedFromFee;


    mapping (address => bool) private _isExcluded;

    address[] private _excluded;


    uint256 private constant MAX = ~uint256(0);

    uint256 private _tTotal = 1_000_000_000_000_000 * 10**18;

    uint256 private _rTotal = (MAX - (MAX % _tTotal));

    uint256 private _tFeeTotal;


    string private _name = "Frontline Heroes Token";

    string private _symbol = "FHT";

    uint8 private _decimals = 18;


    uint256 public _taxFee = 2;

    uint256 private _previousTaxFee = _taxFee;


    uint256 public _liquidityFee = 1;

    uint256 private _previousLiquidityFee = _liquidityFee;


    uint256 public _buybackFee = 2;

    uint256 private _previousBuybackFee = _buybackFee;


    uint256 public  _charityFee = 3;
```

```
uint256 private _previousCharityFee = _charityFee;

uint256 public _maxTxAmount = 1_000_000_000_000_0 * 10**18;//1%
uint256 private minimumTokensBeforeSwap = 1_000_000_0 * 10**18;

IUniswapV2Router02 public immutable uniswapV2Router;
address public immutable uniswapV2Pair;

bool inSwapAndLiquify;
bool public swapAndLiquifyEnabled = false;
event RewardLiquidityProviders(uint256 tokenAmount);
event SwapAndLiquifyEnabledUpdated(bool enabled);
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
    uint256 tokensIntoLiqudity
);


event SwapTokensForETH(
    uint256 amountIn,
    address[] path
);

modifier lockTheSwap {
    inSwapAndLiquify = true;
    _;
    inSwapAndLiquify = false;
}

constructor()
```

```
    {
        _rOwned[_msgSender()] = _rTotal;
        IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
        uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this), _uniswapV2Router.WETH());
        uniswapV2Router = _uniswapV2Router;
        _isExcludedFromFee[owner()] = true;
        _isExcludedFromFee[address(this)] = true;
        emit Transfer(address(0), _msgSender(), _tTotal);
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function totalSupply() public view override returns (uint256) {
        return _tTotal;
    }

    function balanceOf(address account) public view override returns (uint256) {
        if (_isExcluded[account]) return _tOwned[account];
        return tokenFromReflection(_rOwned[account]);
```

```
    }

    function transfer(address recipient, uint256 amount) public override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender) public view override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns
(bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "BEP20: transfer
amount exceeds allowance"));
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue,
"BEP20: decreased allowance below zero"));
```

```
        return true;
    }

    function isExcludedFromReward(address account) public view returns (bool) {
        return _isExcluded[account];
    }

    function totalFees() public view returns (uint256) {
        return _tFeeTotal;
    }

    function minimumTokensBeforeSwapAmount() public view returns (uint256) {
        return minimumTokensBeforeSwap;
    }


    function deliver(uint256 tAmount) public {
        address sender = _msgSender();
        require(!_isExcluded[sender], "Excluded addresses cannot call this function");
        (uint256 rAmount,,,,,) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rTotal = _rTotal.sub(rAmount);
        _tFeeTotal = _tFeeTotal.add(tAmount);
    }


    function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view
returns(uint256) {
        require(tAmount <= _tTotal, "Amount must be less than supply");
        if (!deductTransferFee) {
            (uint256 rAmount,,,,,) = _getValues(tAmount);
```

```
            return rAmount;
        } else {
            (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
            return rTransferAmount;
        }
    }

    function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
        require(rAmount <= _rTotal, "Amount must be less than total reflections");
        uint256 currentRate =  _getRate();
        return rAmount.div(currentRate);
    }

    function excludeFromReward(address account) public onlyOwner() {

        require(!_isExcluded[account], "Account is already excluded");
        if(_rOwned[account] > 0) {
            _tOwned[account] = tokenFromReflection(_rOwned[account]);
        }
        _isExcluded[account] = true;
        _excluded.push(account);
    }

    function includeInReward(address account) external onlyOwner() {
        require(_isExcluded[account], "Account is already excluded");
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_excluded[i] == account) {
                _excluded[i] = _excluded[_excluded.length - 1];
                _tOwned[account] = 0;
                _isExcluded[account] = false;
                _excluded.pop();
```

```
                break;
            }
        }
    }

    function _approve(address owner, address spender, uint256 amount) private {
        require(owner != address(0), "BEP20: approve from the zero address");
        require(spender != address(0), "BEP20: approve to the zero address");

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

    function _transfer(address from, address to, uint256 amount) private
    {
        require(from != address(0), "BEP20: transfer from the zero address");
        require(to != address(0), "BEP20: transfer to the zero address");
        require(amount > 0, "Transfer amount must be greater than zero");

        if(from != owner() && to != owner())
        {
            require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");
        }

        if (!inSwapAndLiquify && swapAndLiquifyEnabled &&  from != uniswapV2Pair && from !=
owner())
        {
            uint256 contractTokenBalance = balanceOf(address(this));
            if (contractTokenBalance >= minimumTokensBeforeSwap)
            {
                contractTokenBalance = minimumTokensBeforeSwap;
```

```
            swapAndLiquify(contractTokenBalance);
        }
        checkForBuyBack();
    }

    bool takeFee = true;
    //if any account belongs to _isExcludedFromFee account then remove the fee
    if(_isExcludedFromFee[from] || _isExcludedFromFee[to]){
        takeFee = false;
    }
    _tokenTransfer(from,to,amount,takeFee);
}

function swapAndLiquify(uint256 toksns) private lockTheSwap
{
    uint256 totalFee = _liquidityFee.add(_buybackFee).add(_charityFee);
    uint256 halfLiquidityTokens = toksns.mul(_liquidityFee).div(totalFee).div(2);
    uint256 swapableTokens = toksns.sub(halfLiquidityTokens);
    uint256 initialBalance = address(this).balance;
    swapTokensForEth(swapableTokens);
    uint256 newBalance = address(this).balance.sub(initialBalance);
    uint256 bnbForLiquidity = newBalance.mul(_liquidityFee).div(totalFee).div(2);
    addLiquidity(halfLiquidityTokens, bnbForLiquidity);
    uint256 bnbForcharity = newBalance.mul(_charityFee).div(totalFee);
    transferToAddressETH(charityWalletAddress, bnbForcharity);
    //remaining balance left for buyback.
}


function swapTokensForEth(uint256 tokenAmount) private {
    // generate the uniswap pair path of token -> weth
```

```
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        _approve(address(this), address(uniswapV2Router), tokenAmount);
        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount, 0,
path, address(this), block.timestamp);
        emit SwapTokensForETH(tokenAmount, path);
    }


    function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router), tokenAmount);
        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this), tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            owner(), block.timestamp
        );
    }


    function _tokenTransfer(address sender, address recipient, uint256 amount,bool takeFee) private {
        if(!takeFee)
            removeAllFee();

        if (_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferFromExcluded(sender, recipient, amount);
        } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
            _transferToExcluded(sender, recipient, amount);
        } else if (_isExcluded[sender] && _isExcluded[recipient]) {
```

```
            _transferBothExcluded(sender, recipient, amount);
        } else {
            _transferStandard(sender, recipient, amount);
        }

        if(!takeFee)
            restoreAllFee();
    }

    function _transferStandard(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256
tFee, uint256 tLiquidity) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256
tFee, uint256 tLiquidity) = _getValues(tAmount);
            _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private {
```

```
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256
tFee, uint256 tLiquidity) = _getValues(tAmount);
            _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256
tFee, uint256 tLiquidity) = _getValues(tAmount);
            _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _takeLiquidity(tLiquidity);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    function _reflectFee(uint256 rFee, uint256 tFee) private {
        _rTotal = _rTotal.sub(rFee);
        _tFeeTotal = _tFeeTotal.add(tFee);
    }

    function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256,
uint256, uint256) {
        (uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = _getTValues(tAmount);
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFee,
tLiquidity, _getRate());
```

```
        return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tLiquidity);
    }

    function _getTValues(uint256 tAmount) private view returns (uint256, uint256, uint256) {
        uint256 tFee = calculateTaxFee(tAmount);
        uint256 tLiquidity = calculateLiquidityFee(tAmount);
        uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity);
        return (tTransferAmount, tFee, tLiquidity);
    }

    function _getRValues(uint256 tAmount, uint256 tFee, uint256 tLiquidity, uint256 currentRate) private
pure returns (uint256, uint256, uint256) {
        uint256 rAmount = tAmount.mul(currentRate);
        uint256 rFee = tFee.mul(currentRate);
        uint256 rLiquidity = tLiquidity.mul(currentRate);
        uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity);
        return (rAmount, rTransferAmount, rFee);
    }

    function _getRate() private view returns(uint256) {
        (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
        return rSupply.div(tSupply);
    }

    function _getCurrentSupply() private view returns(uint256, uint256) {
        uint256 rSupply = _rTotal;
        uint256 tSupply = _tTotal;
        for (uint256 i = 0; i < _excluded.length; i++) {
            if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal,
_tTotal);
            rSupply = rSupply.sub(_rOwned[_excluded[i]]);
            tSupply = tSupply.sub(_tOwned[_excluded[i]]);
```

```
    }
    if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
    return (rSupply, tSupply);
}

function _takeLiquidity(uint256 tLiquidity) private {
    uint256 currentRate =  _getRate();
    uint256 rLiquidity = tLiquidity.mul(currentRate);
    _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity);
    if(_isExcluded[address(this)])
        _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity);
}

function calculateTaxFee(uint256 _amount) private view returns (uint256) {
    return _amount.mul(_taxFee).div(
        10**2
    );
}

function calculateLiquidityFee(uint256 _amount) private view returns (uint256) {
    return _amount.mul(_liquidityFee.add(_buybackFee).add(_charityFee)).div(100);
}

function removeAllFee() private {
    _taxFee = 0;
    _liquidityFee = 0;
    _buybackFee = 0;
    _charityFee = 0;
}

function restoreAllFee() private {
```

```
        _taxFee = _previousTaxFee;
        _liquidityFee = _previousLiquidityFee;
        _buybackFee = _previousBuybackFee;
        _charityFee = _previousCharityFee;
    }


    function prepareForPreSale() external onlyOwner {
        setSwapAndLiquifyEnabled(false);
        _maxTxAmount = _tTotal;
        _taxFee = 0;
        _previousTaxFee = _taxFee;
        _liquidityFee = 0;
        _previousLiquidityFee = _liquidityFee;
        _buybackFee = 0;
        _previousBuybackFee = _buybackFee;
        _charityFee = 0;
        _previousCharityFee = _charityFee;
    }

    function afterPreSale() external onlyOwner {
        setSwapAndLiquifyEnabled(true);
        _maxTxAmount = _tTotal.div(100);
        _taxFee = 2;
        _previousTaxFee = _taxFee;
        _liquidityFee = 1;
        _previousLiquidityFee = _liquidityFee;
        _buybackFee = 2;
        _previousBuybackFee = _buybackFee;
        _charityFee = 3;
        _previousCharityFee = _charityFee;
```

```
}



function isExcludedFromFee(address account) public view returns(bool) {
    return _isExcludedFromFee[account];
}

function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}

function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    _taxFee = taxFee;
}

function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
    _liquidityFee = liquidityFee;
}

function setBuybackFeePercent(uint256 buybackFee) external onlyOwner() {
    _buybackFee = buybackFee;
}

function setCharityFeePercent(uint256 charityFee) external onlyOwner() {
    _charityFee = charityFee;
}
```

```
    function setMaxTxAmount(uint256 maxTxAmount) external onlyOwner() {
        _maxTxAmount = maxTxAmount;
    }

    function setNumTokensSellToAddToLiquidity(uint256 _minimumTokensBeforeSwap) external
onlyOwner() {
        minimumTokensBeforeSwap = _minimumTokensBeforeSwap;
    }


    function setcharityWalletAddress(address _charityWalletAddress) external onlyOwner() {
        charityWalletAddress = payable(_charityWalletAddress);
    }

    function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
        swapAndLiquifyEnabled = _enabled;
        emit SwapAndLiquifyEnabledUpdated(_enabled);
    }



    function transferToAddressETH(address payable recipient, uint256 amount) private {
        recipient.transfer(amount);
    }

     //to recieve ETH from uniswapV2Router when swaping
    receive() external payable {}

/////---BUYBACK----////
```

```
event SwapETHForTokens(uint256 amountIn, address[] path);
uint256 private buyBackUpperLimit = 1 * 10**18;
uint256 private minBalanceForBuyback = 1 * 10**18;
uint256 private buybackFactor = 1;
bool public buyBackEnabled = true;
event BuyBackEnabledUpdated(bool enabled);

function buyBackUpperLimitAmount() public view returns (uint256) {
    return buyBackUpperLimit;
}

function buyBackTokens(uint256 amount) private lockTheSwap
{
        if (amount > 0) {
            swapETHForTokens(amount);
            }
}

function checkForBuyBack() private lockTheSwap
{
    uint256 balance = address(this).balance;
    if (buyBackEnabled && balance > minBalanceForBuyback)
    {
        if (balance > buyBackUpperLimit)
        {
            balance = buyBackUpperLimit;
        }
        buyBackTokens(balance.div(100).mul(buybackFactor));
    }
}
```

```
function swapETHForTokens(uint256 amount) private
{
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = uniswapV2Router.WETH();
    path[1] = address(this);
  // make the swap
    uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(
        0, // accept any amount of Tokens
        path, deadAddress, // Burn address
        block.timestamp.add(300));
    emit SwapETHForTokens(amount, path);
}

 function setBuybackUpperLimit(uint256 buyBackLimit) external onlyOwner() {
    buyBackUpperLimit = buyBackLimit * 10**18;
}

 function setMinBalanceForBuyback(uint256 _balanceInWei) external onlyOwner() {
    minBalanceForBuyback = _balanceInWei;
}

 function setBuybackFactor(uint256 _percent) external onlyOwner() {
    buybackFactor = _percent;
}

function setBuyBackEnabled(bool _enabled) public onlyOwner {
    buyBackEnabled = _enabled;
    emit BuyBackEnabledUpdated(_enabled);
}
```

```
    function manualBuyback(uint256 amountWei) external onlyOwner()
    {
        buyBackTokens(amountWei);
    }

}
```